DOCUMENT RESUME

ED 084 155 SE 016 995

AUTHOR Huff, George A.
TITLE Geometry and Formal Linguistics.
INSTITUTION Stanford Univ., Calif. Inst. for Mathematical Studies in Social Science.
SPONS AGENCY National Science Foundation, Washington, D.C.
REPORT NO TR-201
PUB DATE Apr 73
NOTE 65p.; Psychology and Education Series

EDRS PRICE MF-$0.65 HC-$3.29
DESCRIPTORS Concept Formation; *Geometry; Learning; *Linguistics; Mathematical Concepts; *Mathematics; Research; *Structural Analysis; *Structural Linguistics; Topology
IDENTIFIERS Proof (Mathematics)

ABSTRACT

This paper presents a method of encoding geometric line-drawings in a way which allows sets of such drawings to be interpreted as formal languages. A characterization of certain geometric predicates in terms of their properties as languages is obtained, and techniques usually associated with generative grammars and formal automata are then applied to the geometric framework. Section 1 of the paper specifies the geometric framework; section 2 develops the background material on formal languages, grammars, and automata; section 3 is concerned with geometric predicates; section 4 covers invariance theorems; and section 5 indicates areas for further investigations. (DT)

FILMED FROM BEST AVAILABLE COPY

U.S. DEPARTMENT OF HEALTH, EDUCATION & WELFARE
NATIONAL INSTITUTE OF EDUCATION
THIS DOCUMENT HAS BEEN REPRODUCED EXACTLY AS RECEIVED FROM THE PERSON OR ORGANIZATION ORIGINATING IT. POINTS OF VIEW OR OPINIONS STATED DO NOT NECESSARILY REPRESENT OFFICIAL NATIONAL INSTITUTE OF EDUCATION POSITION OR POLICY

ED 084155

GEOMETRY AND FORMAL LINGUISTICS

BY

GEORGE A. HUFF

TECHNICAL REPORT NO. 201

APRIL 27, 1973

PSYCHOLOGY AND EDUCATION SERIES

INSTITUTE FOR MATHEMATICAL STUDIES IN THE SOCIAL SCIENCES
STANFORD UNIVERSITY
STANFORD, CALIFORNIA

SE 016 995

TECHNICAL REPORTS

PSYCHOLOGY SERIES

INSTITUTE FOR MATHEMATICAL STUDIES IN THE SOCIAL SCIENCES

(Place of publication shown in parentheses; if published title is different from title of Technical Report, this is also shown in parentheses.)

(For reports no. 1 - 44, see Technical Report no. 125.)

50 R. C. Atkinson and R. C. Calfee. Mathematical learning theory. January 2, 1963. (In B. B. Wolman (Ed.), Scientific Psychology. New York: Basic Books, Inc., 1965. Pp. 254-275)

51 P. Suppes, E. Crothers, and R. Weir. Application of mathematical learning theory and linguistic analysis to vowel phoneme matching in Russian words. December 28, 1962.

52 R. C. Atkinson, R. Calfee, G. Sommer, W. Jeffrey and R. Shoemaker. A test of three models for stimulus compounding with children. January 29, 1963. (J. exp. Psychol., 1964, 67, 52-58)

53 E. Crothers. General Markov models for learning with inter-trial forgetting. April 8, 1963.

54 J. L. Myers and R. C. Atkinson. Choice behavior and reward structure. May 24, 1963. (Journal math. Psychol., 1964, 1, 170-203)

55 R. E. Robinson. A set-theoretical approach to empirical meaningfulness of measurement statements. June 10, 1963.

56 E. Crothers, R. Weir and P. Palmer. The role of transcription in the learning of the orthographic representations of Russian sounds. June 17,. 1963.

57 P. Suppes. Problems of optimization in learning a list of simple items. July 22, 1963. (In Maynard W. Shelly, II and Glenn L. Bryan (Eds.), Human Judgments and Optimality. New York: Wiley. 1964. Pp. 116-126)

58 R. C. Atkinson and E. J. Crothers. Theoretical note: all-or-none learning and intertrial forgetting. July 24, 1963.

59 R. C. Calfee. Long-term behavior of rats under probabilistic reinforcement schedules. October 1, 1963.

60 R. C. Atkinson and E. J. Crothers. Tests of acquisition and retention, axioms for paired-associate learning. October 25, 1963. (A comparison of paired-associate learning models having different acquisition and retention axioms, J. math. Psychol., 1964, 1, 285-315)

61 W. J. McGill and J. Gibbon. The general-gamma distribution and reaction times. November 20, 1963. (J. math. Psychol., 1965, 2, 1-18)

62 M. F. Norman. Incremental learning on random trials. December 9, 1963. (J. math. Psychol., 1964, 1, 336-351)

63 P. Suppes. The development of mathematical concepts in children. February 25, 1964. (On the behavioral foundations of mathematical concepts. Monographs of the Society for Research in Child Development, 1965, 30, 60-96)

64 P. Suppes. Mathematical concept formation in children. April 10, 1964. (Amer. Psychologist, 1966, 21, 139-150)

65 R. C. Calfee, R. C. Atkinson, and T. Shelton, Jr. Mathematical models for verbal learning. August 21, 1964. (In N. Wiener and J. P. Schoda (Eds.), Cybernetics of the Nervous System: Progress in Brain Research. Amsterdam, The Netherlands: Elsevier Publishing Co., 1965. Pp. 333-349)

66 L. Keller, M. Cole, C. J. Burke, and W. K. Estes. Paired associate learning with differential rewards. August 20, 1964. (Reward and information values of trial outcomes in paired associate learning. (Psychol. Monogr., 1965, 79, 1-21)

67 M. F. Norman. A probabilistic model for free-responding. December 14, 1964.

68 W. K. Estes and H. A. Taylor. Visual detection in relation to display size and redundancy of critical elements. January 25, 1965, Revised 7-1-65. (Perception and Psychophysics, 1966, 1, 9-16)

69 P. Suppes and J. Donio. Foundations of stimulus-sampling theory for continuous-time processes. February 9, 1965. (J. math. Psychol., 1967, 4, 202-225)

70 R. C. Atkinson and R. A. Kinchla. A learning model for forced-choice detection experiments. February 10, 1965. (Br. J. math stat. Psychol., 1965, 18, 184-206)

71 E. J. Crothers. Presentation orders for items from different categories. March 10, 1965.

72 P. Suppes, G. Groen, and M. Schlag-Rey. Some models for response latency in paired-associates learning. May 5, 1965. (J. math. Psychol., 1966, 3, 99-128)

73 M. V. Levine. The generalization function in the probability learning experiment. June 3, 1965.

74 D. Hansen and T. S. Rodgers. An exploration of psycholinguistic units in initial reading. July 6, 1965.

75 B. C. Arnold. A correlated urn-scheme for a continuum of responses. July 20, 1965.

76 C. Izawa and W. K. Estes. Reinforcement-test sequences in paired-associate learning. August 1, 1965. (Psychol. Reports, 1966, 18, 879-919)

77 S. L. Filehart. Pattern discrimination learning with Rhesus monkeys. September 1, 1965. (Psychol. Reports, 1966, 19, 311-324)

78 J. L. Phillips and R. C. Atkinson. The effects of display size on short-term memory. August 31, 1965.

79 R. C. Atkinson and R. M. Shiffrin. Mathematical models for memory and learning. September 20, 1965.

80 P. Suppes. The psychological foundations of mathematics. October 25, 1965. (Colloques Internationaux du Centre National de la Recherche Scientifique. Editions du Centre National de la Recherche Scientifique. Paris: 1967. Pp. 213-242)

81 P. Suppes. Computer-assisted instruction in the schools: potentialities, problems, prospects. October 29, 1965.

82 R. A. Kinchla, J. Townsend, J. Yellott, Jr., and R. C. Atkinson. Influence of correlated visual cues on auditory signal detection. November 2, 1965. (Perception and Psychophysics, 1966, 1, 67-73)

83 P. Suppes, M. Jerman, and G. Groen. Arithmetic drills and review on a computer-based teletype. November 5, 1965. (Arithmetic Teacher, April 1966, 303-309.

84 P. Suppes and L. Hyman. Concept learning with non-verbal geometrical stimuli. November 15, 1965.

85 P. Holland. A variation on the minimum chi-square test. (J. math. Psychol., 1967, 3, 377-413).

86 P. Suppes. Accelerated program in elementary-school mathematics -- the second year. November 22, 1965. (Psychology in the Schools, 1966, 3, 294-307)

87 P. Lorenzen and F. Binford. Logic as a dialogical game. November 29, 1965.

88 L. Keller, W. J. Thomson, J. R. Tweedy, and R. C. Atkinson. The effects of reinforcement interval on the acquisition of paired-associate responses. December 10, 1965. (J. exp. Psychol., 1967, 73, 268-277)

89 J. I. Yellott, Jr. Some effects on noncontingent success in human probability learning. December 15, 1965.

90 P. Suppes and G. Groen. Some counting models for first-grade performance data on simple addition facts. January 14, 1966. (In J. M. Scandura (Ed.), Research in Mathematics Education. Washington, D. C.: NCTM, 1967. Pp. 35-43.

91 P. Suppes. Information processing and choice behavior. January 31, 1966.

92 G. Groen and R. C. Atkinson. Models for optimizing the learning process. February 11, 1966. (Psychol. Bulletin, 1966, 66, 309-320)

93 R. C. Atkinson and D. Hansen. Computer-assisted instruction in initial reading: Stanford project. March 17, 1966. (Reading Research Quarterly, 1966, 2, 5-25)

94 P. Suppes. Probabilistic inference and the concept of total evidence. March 23, 1966. (In J. Hintikka and P. Suppes (Eds.), Aspects of Inductive Logic. Amsterdam: North-Holland Publishing Co., 1966. Pp. 49-65.

95 P. Suppes. The axiomatic method in high-school mathematics. April 12, 1966. (The Role of Axiomatics and Problem Solving in Mathematics. The Conference Board of the Mathematical Sciences, Washington, D. C. Ginn and Co., 1966. Pp. 69-76.

(Continued on inside back cover)

ED 084155

GEOMETRY AND FORMAL LINGUISTICS

by

George A. Huff

TECHNICAL REPORT NO. 201

April 27, 1973

PSYCHOLOGY AND EDUCATION SERIES

Reproduction in Whole or in Part Is Permitted for

Any Purpose of the United States Government

Copyright © 1973, by George A. Huff
All rights reserved

"PERMISSION TO REPRODUCE THIS COPYRIGHTED MATERIAL HAS BEEN GRANTED BY

George A. Huff

TO ERIC AND ORGANIZATIONS OPERATING UNDER AGREEMENTS WITH THE NATIONAL INSTITUTE OF EDUCATION. FURTHER REPRODUCTION OUTSIDE THE ERIC SYSTEM REQUIRES PERMISSION OF THE COPYRIGHT OWNER."

INSTITUTE FOR MATHEMATICAL STUDIES IN THE SOCIAL SCIENCES

STANFORD UNIVERSITY

STANFORD, CALIFORNIA

TABLE OF CONTENTS

ACKNOWLEDGMENTS	i
INTRODUCTION	1
SECTION 1: GEOMETRIC FRAMEWORK	3
SECTION 2: GRAMMARS AND AUTOMATA	13
SECTION 3: GEOMETRIC PREDICATES	19
SECTION 4: INVARIANCE THEOREMS	39
SECTION 5: FURTHER INVESTIGATIONS	51
BIBLIOGRAPHY	58

ACKNOWLEDGMENTS

I wish to express my sincere gratitude to all those through whose hands this dissertation passed prior to its completion, and particularly, to Professor Patrick Suppes, whose encouragement and patience over a long period of time deserves special mention.

Partial financial support was received from the National Science Foundation, Grant NSFGJ-443X3.

INTRODUCTION

In this paper we will present a method of encoding geometric line-drawings in a way which allows sets of such drawings to be interpreted as formal languages, the purpose being to obtain a characterization of certain geometric predicates in terms of their properties as languages. Techniques usually associated with generative grammars and formal automata can then be applied to this geometric framework.

By way of background, it should be mentioned that the results contained herein are an extension of work begun by William Rottmayer and myself in the summer of 1969, under the direction of Professor Patrick Suppes. Our effort was then informally described as a study of geometric concept formation, as it might be applied to the learning of such concepts by children. The formulation presented here is an outgrowth of that summer's work. More extensive background material appears in Rottmayer's report, "A Formal Theory of Perception."

Our investigation had been motivated primarily by a study of perceptrons, which are formal machines capable of learning to recognise certain classes of geometric figures. A perceptron accepts input in the form of figures represented on a checkerboard-like grid, and through a series of reinforced trials, its internal coefficients are modified to converge to a state where recognition is perfect. The ability of a perceptron to learn to recognise even the simplest of geometric predicates is extremely limited, as demonstrated by Minsky and Papert in Perceptrons. Another framework for the kind of learning which can be applied to geometric concepts is suggested by Professor Suppes' paper "Stimulus-Response Theory of Finite Automata." This paper indicates that the

stimulus-response theory of learning can be adequately applied to the learning of any task which can be performed by a finite automaton; that is, for any such task, the finite automaton which performs it represents the convergent state of a stimulus-response learning model for that task. In view of the possibility that this result could be extended to include stronger forms of automata, in particular, various kinds of Turing machines, our goal in this paper is to classify certain geometric predicates on the basis of their recognition-potential by the various classes of automata.

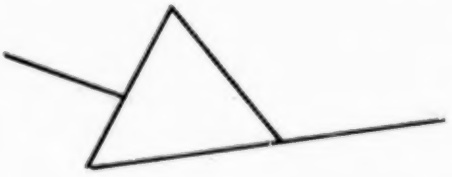
SECTION 1: GEOMETRIC FRAMEWORK

The first problem is to specify the kinds of geometric figures which most naturally lend themselves to a simple learning situation. It should be noted that, whereas the perceptron was intented to be a general pattern recognition device with no particular emphasis on geometric aspects, we have chosen to focus on those geometric aspects themselves. Roughly speaking, the kind of geometry we want to consider is that portion of Euclidean geometry consisting of all finite straight-line drawings in the plane: that is, we take for our domain all planar figures made up of a finite number of line segments, each of a finite length. We will identify each predicate defined over a domain of such figures with its extension as a set of line-drawings. Hence a figure F can be said to satisfy the predicate "contains a triangle" if and only if $F \in \{x | x \text{ is a figure containing a triangle}\}$. We want to exclude predicates which depend on the quantitative aspects of these drawings, such as the exact measurement of area, length or angle, as well as those predicates which are a function of the orientation of the drawings, and so on. For example, we would like to include such predicates as "is a triangle" and "contains a triangle" while excluding "is an equilateral triangle" and "is a horizontal line." By committing ourselves to such restrictions on our figure domain we isolate a class of figures which is moderately rich in geometric properties and yet lends itself well to linguistic treatment.

It seems natural then that the encoding of a figure should convey information about the lines present and how they intersect one another. Certainly the translation of a particular instance of a figure into such

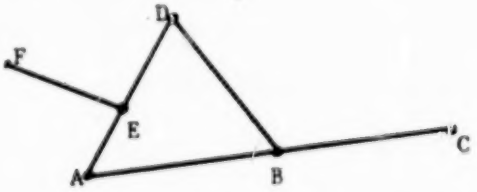
a representation as a coded set of line segments will represent a considerable abstraction. For example, any two triangles would be coded identically, whether they be isosceles, equilateral, or whatever: the coding would show three line segments, each pair of which has a unique endpoint in common. Let us consider how an encoding might be derived for the following figure:

Figure 1



First assign to each vertex, that is, each endpoint of a line segment or intersection of line segments, a unique Roman letter, as in Figure 2:

Figure 2



Each line segment in the figure will be encoded as an ordered sequence of those letters which correspond to the vertices on the line segment. Thus ABC tells us that one line segment in the figure has endpoints A and C and central vertex B. A complete coding for the figure is a list of all the line segments that appear, e.g. ABC, BD, DEA, and FE. It should be evident that there is such a coding for every figure, whether the figure is in the form of a line drawing as given here or a set of Cartesian coordinates in the plane. This coding is not unique; the most obvious reason is that the vertex letters were assigned arbitrarily. Furthermore, the line segments could have been listed in a different order and the vertices within a line segment could have been

listed in reverse order. However this lack of uniqueness will cause no difficulty because there is a simple algorithm for determining whether or not two codings apply to the same figure.

The most natural way to formalize this coding procedure is to represent a coding for a figure as a set of codings for its line segments, relative to a given labelling of its vertices. Each line segment would be encoded by its vertex-labels, listed in order from endpoint to endpoint. Thus the above example would become {ABC, BD, DEA, FE}. Other codings for the same figure are {DB, FE, DEA, CBA} and {KXY, AX, KHA, WH}. For our initial discussion of codings we will use this set-theoretic notation, as its primary advantage is convenience. We soon realize though that the usual benefits associated with set theory do not accrue: we can use unions and intersections of codings only with extreme care, and it is not at all clear what the complement of a coding or set of codings should be.

We can now give a more precise definition of what we shall mean throughout this paper by the word "coding." Let $\mathcal{A} = \{A_1, A_2, \ldots A_n, \ldots\}$ be a countably infinite set: the A_i's will be vertex-labels, and we will use A, B, C, ... as variables ranging over the set $\mathcal{A}$.

Definition 1.1 A line segment ls is a finite list of elements from $\mathcal{A}$ of length at least two and such that no A_i occurs more than once.[1]

By way of notation we let Λ denote the set of all such line segments and [ls] the set of all vertex-labels occurring in a line segment ls.

[1]Clearly we really mean to say "a coding for a line segment ls ..." but for convenience we will shorten this to "a line segment ls ..." when no confusion will arise.

The length of ls is denoted $|ls|$ and is equal to $\overline{[ls]}$. An endpoint of ls is a vertex-label which occurs first or last in the list ls.

Definition 1.2 A coding c is a finite set of line segments such that: first, if $ls_1 \in c$ and $ls_2 \in c$ then $\overline{[ls_1] \cap [ls_2]} < 2$, and second, if $ls \in c$ and $A \in [ls]$ then either A is an endpoint of ls or there is an $ls' \in c$ with $A \in [ls']$ and $ls' \neq ls$.

We let $\mathbf{C}$ be the set of all codings. The conditions in Definition 1.2 require that no pair of line segments can intersect more than once and that each vertex represent an endpoint, a genuine point of intersection, or both. To sort out the many different codings which would correspond to the same figures, we give the following:

Definition 1.3 Let $\equiv$ be a binary relation on $\mathbf{C}$ defined as follows: if c_1 and c_2 are codings then $c_1 \equiv c_2$ if and only if there is a one-to-one function f mapping $\bigcup_{ls \in c_1} [ls]$ onto $\bigcup_{ls \in c_2} [ls]$ such that

1) if $ls \in c_1$ and $ls = A_{n_1}A_{n_2}\ldots A_{n_i}$ then either $f(A_{n_1})f(A_{n_2})\ldots f(A_{n_i}) \in c_2$ or $f(A_{n_i})f(A_{n_{i-1}})\ldots f(A_{n_1}) \in c_2$, and conversely,

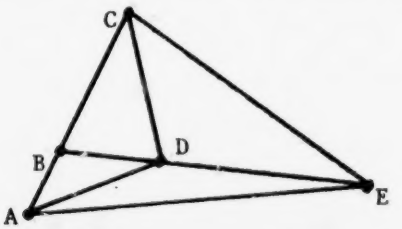
2) if $ls \in c_2$ and $ls = A_{n_1}A_{n_2}\ldots A_{n_i}$ then either $g(A_{n_1})g(A_{n_2})\ldots g(A_{n_i}) \in c_1$ or $g(A_{n_i})g(A_{n_{i-1}})\ldots g(A_{n_1}) \in c_1$, where $g = f^{-1}$.

This definition says that two codings are equivalent ("$\equiv$") provided they differ only in the names of the vertex-labels present and in the order in which the line segments are taken. Note that $\bigcup_{ls \in c} [ls]$ is the set of all vertex-labels appearing in the coding c.

As an example, consider the codings c_1 = {ABC, BDE, AE, AD, CE, CD}

and c_2 = {DEF, EAB, DB, BF, AF, AD}, both derived from the figure:

Figure 3



Then $\bigcup_{ls \in c_1}$ [ls] = [ABC] ∪ [BDE] ∪ [AE] ∪ [AD] ∪ [CE] ∪ [CD] = {A, B, C, D, E} and similarly $\bigcup_{ls \in c_2}$ [ls] = {A, B, D, E, F}. To see that $c_1 \equiv c_2$, let f: {A, B, C, D, E} → {A, B, D, E, F} be defined by f(A) = D, f(B) = E, f(C) = F, f(D) = A, f(E) = B. The line segment ABC in c_1 then corresponds to the line segment f(A)f(B)f(C) = DEF in c_2.

It is clear from the nature of the definition that "≡" is an equivalence relation, and hence partitions $\mathcal{C}$ into disjoint classes of equivalent codings. An obvious theorem relating equivalence for codings to figures in the plane is this:

Theorem 1.4 If c_1 is a coding for a figure F and $c_1 \equiv c_2$, then c_2 is also a coding for F. Conversely, if c_1 and c_2 are codings for F then $c_1 \equiv c_2$. Thus each figure determines a unique class of codings.

There are two important limitations of the codings as defined above. First, there are a number of geometric predicates which are inexpressible, in addition to those more or less quantitative predicates which we specifically excluded earlier. Generally speaking, these are predicates which are dependent upon the preservation of inside-outside relationships within a class of equivalent codings. For example, the drawing in Figure 2 would have the same (that is, an equivalent) coding as the following figure, in which vertex F has been shifted to the interior

of the triangle:

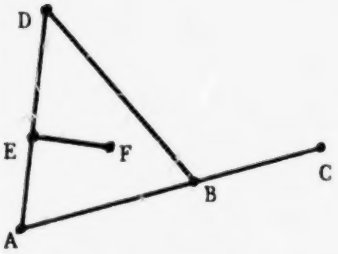


Figure 4

A special case of this type of problem is convexity: we can tell from its coding whether or not a figure is a polygon, but there is no way to distinguish convex polygons from concave ones. Another example to show the extent to which deformations within a class of figures can affect the notion of interior and exterior is this:

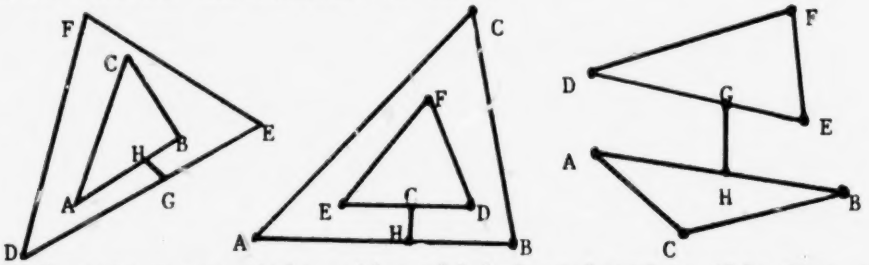


Figure 5

A most important geometric notion which is related to this type of problem concerns the identification of the regions, or faces,[2] into which a figure divides the plane. For example the following two figures have the same coding:

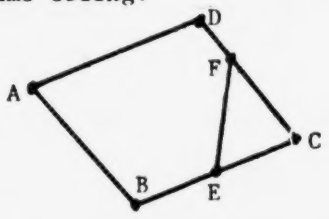


Figure 6

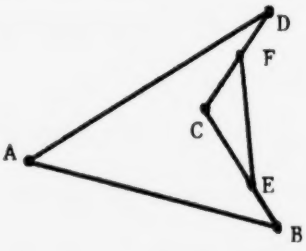


Figure 7

Figure 6 has faces bounded by ABEFD and CEF whereas Figure 7 has faces bounded by ABCD and CEF. Although we have no means of explicitly identifying the vertices which bound the faces of a figure, it is a simple

[2]A "face" is usually considered to be an area of the plane which is bounded by lines of a figure and which contains neither lines nor vertices in its interior. Every finite figure determines a unique infinite face, but we will follow convention and disregard it.

matter to show that the number of faces will remain constant within a class of figures having equivalent codings. To do this we use Euler's formula[3], relating the number of vertices and line segments present in a figure to the number of faces:

$$V - LS + F = N$$

where V is the number of vertices , LS is the number of line segments (where a line segment coded as ABCD is considered to be comprised of the three line segments AB, BC and CD), F is the number of bounded faces, and N is the number of connected components making up the figure. In the previous example (Figures 6 and 7) V = 6, LS = 7 and N = 1, hence F = 1 + 7 - 6 = 2. For a given equivalence class of codings, the number of vertices and line segments is clearly invariant, the latter being given by $\sum_{lsec}([ls] - 1)$. Also, the number of connected components can be determined from any coding by first grouping vertices into classes according to whether they are connected to one another by a path of line segments, and then counting these classes. Hence the number of faces F can be determined as well, and is thus invariant.

The second limitation of our present formulation has to do with the connection between codings and figures in the plane. We have said so far only that any figure determines a class of codings, but unfortunately it does not always happen that a coding determines a non-empty class of planar figures. In fact, the problem of specifying constraints on our system which will guarantee a correspondence between non-empty classes of codings and figures was studied by Rottmayer [pp. 56-61] and turns out to be extremely difficult. We will discuss some background material here, with the intention of indicating the scope of the problem.

---

[3]For more material on Euler's formula, see Berge, The Theory of Graphs, p. 207 ff.

We can begin by defining a process called segmentation, which we will use to obtain new codings from a given one. This process will permit us to introduce a finite number of additional vertices in a line segment and to "break" the original line segment at each of these new points. Thus a line segment ABCD occurring in a coding c could become AA_1, A_1A_2, ..., A_nB, BCD under segmentation; we call the resulting coding a segmentation of the original coding. The idea of course is to use straight-line figures to approximate drawings containing one or more curved lines.

The process of segmentation allows us to classify non-planar codings into two varieties: one which we will call "geometric" and the other "topological." By a topologically non-planar (TNP) coding we mean a non-planar coding which has no planar segmentation. We use the adjective "topological" because such codings will remain non-planar under the sort of elastic deformations of the plane that are studied in algebraic topology. It will be natural to think of such codings in terms of vertices connected by curved lines, since we can approximate such curves by unlimited segmentation without destroying the non-planar property. There are basically two such figures: Kuratowski proved in 1930 that any TNP figure must contain either one or the other as a subfigure.

The first is the "complete graph over five vertices," that is, the figure which contains only five vertices, with lines connecting each pair of vertices. An attempt to draw this figure will result in a figure with a missing vertex:

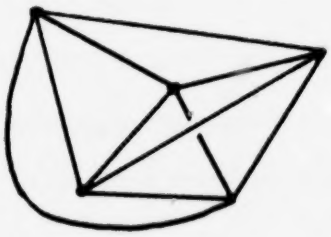
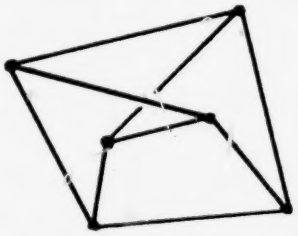


Figure 8

The second consists of two sets of three vertices each, with lines drawn from each vertex in the first set to each vertex in the second:

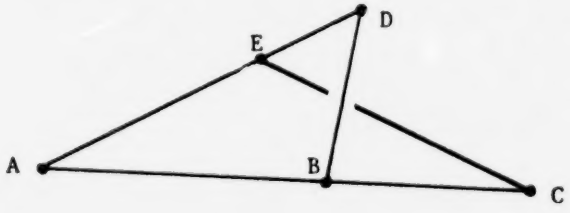
Figure 9



It is worth mentioning the proofs of the non-planarity of these two examples rely on the violation of Euler's formula.

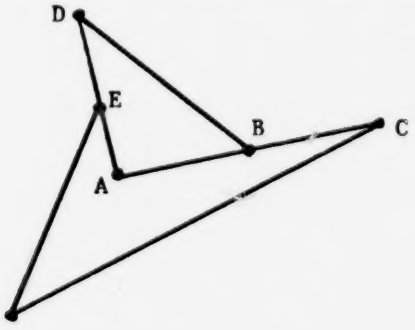
A non-planar coding will be said to be geometrically non-planar (GNP) if it is not TNP, that is, if some segmentation of it is planar. The simplest example of such a coding is c = {ABC, AED, BD, CE}. It would be "drawn" as follows:

Figure 10



Since line segments CE and BD must intersect one another and there is no vertex present in the coding corresponding to this intersection, this "figure" cannot be embedded in the plane. However, by segmenting the line segment CE for example, we get a drawing which is certainly planar:

Figure 11



Another example of a GNP coding is given by c = {ABCD, DEF, CEG, BG, AF}, and would be drawn:

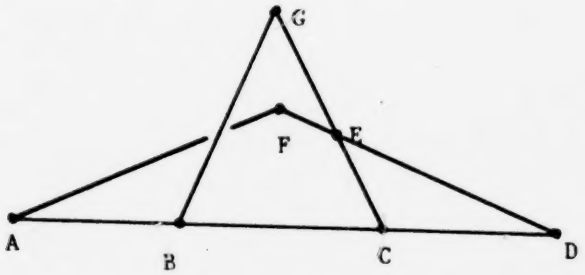


Figure 12

Figures 11 and 12 are distinct in that neither contains the other as a subfigure, nor is there a common GNP subfigure. Obviously, any coding which contains either of these two examples will also be non-planar. These two figures are alike, however, in that they both make use of the fact that a triangle is the only necessarily convex polygon, and hence, when a side is extended, the endpoint must lie outside the triangle. There are other GNP figures though which do not seem to rely on this fact, and hence it seems unlikely that a Kuratowski-type result can be established for the GNP codings.

Although the difficulties posed by these non-planar figures are serious ones, it will still be possible to interpret each coding as a class of plane figures, with possibly missing vertices. None of the simple geometric properties, like connectedness or triangularity, will be affected by this. However, throughout the following, it must be remembered that the correspondence between codings and figures is not an exact one.

SECTION 2: GRAMMARS AND AUTOMATA

In this section we will develop the necessary background material on formal languages, grammars and automata, in order to establish notation and to state the fundamental results which we will need later. Our notation is based on that used in Formal Languages and their Relation to Automata, by Hopcroft and Ullman. This will allow the use of additional theorems appearing there without re-statement.

By an alphabet, or vocabulary, we will mean a finite set V, whose elements can be concatenated to form strings or words. V^* denotes the set of all strings of finite length over the alphabet V. We use the symbol e to denote the empty string and define $V^+ = V^* - \{e\}$.

Definition 2.1 A quadruple $G = \langle V_N, V_T, P, S \rangle$ is a grammar whenever

1) V_N and V_T are non-empty finite disjoint sets. We let $V = V_N \cup V_T$.
2) P is a finite subset of $(V^+ \times V^+) \cup \{(S,e)\}$.
3) $S \in V_N$.

It is customary to refer to V_N, V_T, P, and S as the variables (non-terminals), terminals, production rules and start symbol, respectively, of the grammar G. A production rule $(\alpha,\beta) \in P$ will be represented by $\alpha \to \beta$. Note that we do not allow general rules of the form $\alpha \to e$.

Given a grammar G we can define a binary relation $\to_G$ on V^* as follows: $\alpha \to_G \beta$ if and only if there exist $x \in V^*$ and $y \in V^*$ such that $\alpha = x\lambda y$, $\beta = x\lambda' y$, and $\lambda \to \lambda' \in P$. Thus $\to_G$ represents a one-step

derivation for the grammar G. We can extend $\rightarrow_G$ in the natural way to a new relation $\rightarrow_G^*$ to represent derivations of any finite number of steps, that is, $\alpha \rightarrow_G^* \beta$ if and only if there is an integer $n \geq 0$ and $\alpha_1, \alpha_2, \ldots, \alpha_n \in V^*$ such that $\alpha \rightarrow_G \alpha_1$, $\alpha_1 \rightarrow_G \alpha_2$, $\ldots$, $\alpha_{n-1} \rightarrow_G \alpha_n = \beta$. By the language generated by G, denoted L(G), we will mean the set $\{\alpha | \alpha \in V_T^+ \text{ and } S \rightarrow_G^* \alpha\}$. One stipulation is made regarding the rule $S \rightarrow e$: If this rule occurs in P, then the only derivation in which it may be used is the one-step derivation $S \rightarrow_G^* e$. We will say that two grammars G_1 and G_2 are equivalent if and only if $L(G_1) = L(G_2)$.

We can classify grammars according to the form of their production rules. The most general type of grammar, having no restrictions on the production rules, is that which is defined above, called a Type 0 grammar. If all the production rules of a grammar have the form $S \rightarrow e$ or $\alpha \rightarrow \beta$ where $|\alpha| \leq |\beta|$, the grammar is said to be Type 1, or context-sensitive. A grammar whose rules are of the form $A \rightarrow \beta$ where $A \in V_N$ will be called a Type 2 or context-free grammar. A Type 3 or regular grammar is one whose rules have the form $S \rightarrow e$, $A \rightarrow v$, or $A \rightarrow vB$, where S, A, $B \in V_N$ and $v \in V_T$. It follows that any Type i grammar (i = 1, 2, 3) is also of Type (i-1); furthermore, grammars of different types may be equivalent. The language generated by a Type i grammar will be called a Type i (or context-sensitive, context-free, regular) language.

Definition 2.2 A finite automaton is a quintuple $\mathcal{F} = \langle K, \Sigma, \delta, q_o, F \rangle$ where

1) K and Σ are finite non-empty disjoint sets (the internal states and input alphabet respectively)
2) $\delta: K \times \Sigma \rightarrow K$ (the transition function)

3) $q_0 \in K$ (the start symbol)

4) $F \subseteq K$ (the set of final states)

A finite automaton processes an input string left to right, changing states with each new input symbol as determined by δ, until the end of the string is reached. The string is said to be accepted by $\mathfrak{F}$ if and only if the last state reached in the processing of the input string belongs to F. As with grammars it is natural to extend δ to δ^*: $K \times \Sigma^* \to K$ by the inductive definition

1) if $v \in \Sigma$, then $\delta^*(q,v) = \delta(q,v)$

2) if $\alpha \in \Sigma^*$, then for $v \in \Sigma$, $\delta^*(q,\alpha v) = \delta(\delta^*(q,\alpha),v)$.

Thus α is accepted by $\mathfrak{F}$ if and only if $\delta^*(q_0,\alpha) \in F$.

An important limitation of any finite automaton is its bounded memory capacity. For example, let $\Sigma = \{a, b\}$ and let $L_1 = \{ba^n | n > 0\}$. A finite automaton processing a string in L_1 can remember only a bounded number of a's from the first half of the string, that number being determined by the number of states; beyond that, the automaton must "loop" before getting to the b, and hence it will lose count. When this happens, the automaton cannot determine whether the number of a's after the b is the same as the number before the b. Thus no finite automaton can accept the language L_1.

The following is a fundamental theorem in the theory of automata.

Theorem 2.3 A language is regular, that is, has a regular grammar, if and only if there is a finite automaton which accepts it.

Context-free languages can be similarly characterized in terms

of the type of automata which accept them. A pushdown automaton is essentially a finite automaton, with left-to-right input processing, which has in addition an unbounded memory in the form of a pushdown stack, operating on a first in-last out basis. Such a machine can accept ("recognise") a language like L_1 defined above: as the machine moves left through the initial string of a's, it can store a symbol for each occurrence; then after passing the b, it can compare each of the following symbols with a symbol in its storage. Thus, while it cannot "count" as such, it is capable of making comparisons. By a non-deterministic pushdown automaton, we mean a pushdown automaton which at some or possibly every stage in its computations has more than one possible move it can make. The theorem for context-free languages is this.

Theorem 2.4 A language is context-free if and only if there is a non-deterministic pushdown automaton which accepts it.

We note that a simple context-free grammar which generates L_1 is given by $G = \langle\{S\}, \{a, b\}, P, S\rangle$ where P consists of the two rules $S \to aba$ and $S \to aSa$.

There are similar theorems for characterizing languages of Type 0 and 1; we will use the notion of a Turing machine for that purpose.

Definition 2.5 A Turing machine is a 6-tuple $T = \langle K, \Sigma, \Gamma, \delta, q_o, F\rangle$ where

1) K and Γ are non-empty finite disjoint sets (the states and the tape symbols of T)
2) $\Sigma \subseteq \Gamma$ (the input symbols)

3) $q_o \in K$ and $F \subseteq K$ (the start state and the set of final states)

4) δ: $K \times \Gamma \rightarrow K \times \Gamma \times$ {Left, Right} (the transition function)

In its usual interpretation, a Turing machine consists of an input tape divided into cells, each capable of holding an element of Γ, and a tape head which can scan the cells of the tape one at a time. The input tape has a leftmost cell but is infinite to the right, and the input is placed initially into the first n cells. The tape head stores the current state symbol and after scanning the contents of a cell, it will change the current state symbol, replace the cell contents with another symbol from Γ, and move one cell to the left or right, all of these actions being governed by the transition function. The machine starts in state q_o with the tape head positioned at the leftmost tape cell. It continues to move through the successive configurations of states and tape symbols, and may come to a halt in one of two ways: either by entering a final state, or by reaching a combination of state and tape symbol for which δ is not defined. It is also possible that the machine will never halt. An input tape will be accepted if and only if the machine processing the tape eventually halts in a final state.

It is convenient to provide a special symbol B for a blank cell, with $B \in \Gamma - \Sigma$ and such that the range of the transition function is restricted to $K \times (\Gamma - \{B\}) \times$ {Left, Right}; that is, blanks can only be removed and cannot be printed. The input string, which is finite in length, would then be followed by an infinite number of blank cells; this eliminates the problem of having to add new cells to the right end of the tape when more cells are needed.

There are many different formulations of Turing machines: non-

deterministic ones, ones having two-way infinite tapes or even two-dimensional tapes, others with two or more distinct tapes and tape heads, each under independent control, and so on. Each of these formulations can be shown to be equivalent to the original description. However, one formulation which is not equivalent is known as a linear-bounded automaton (LBA). This is a non-deterministic Turing machine whose tape length is fixed by the length of the input; in other words, the tape head cannot move beyond those cells which contain the input. We can partially overcome this restriction by changing the form of the input by "padding" the real input with a number of blank cells; this limits us to a length of cells which is a linear function of the real input. It is unknown whether deterministic LBA's accept a smaller class of languages than non-deterministic LBA's.

Theorem 2.6 A language is Type 0 (Type 1) if and only if it is accepted by some Turing machine (LBA).

The results mentioned here establish a hierarchy of automata in parallel with the previous ranking of formal grammars. To emphasize this point, we could have defined finite and pushdown automata as special cases of Turing machines. The former is a Turing machine which moves right only and cannot change the symbols it sees. The latter is similar, but with a memory tape which is accessible only from one end and whose length is bounded by the length of the input. The four central types of automata can thus be seen as progressively weaker versions of a Turing machine.

SECTION 3: GEOMETRIC PREDICATES

We can now translate our discussion of codings from the informal set-theoretical notation used earlier to that of formal languages. The natural way to do this is to interpret a set c = {ABC, BDE, DFG, FA}, for example, as a string of symbols c' = ABC#BDE#DFG#FA, where "#" serves as a line-delimitation symbol[1]. Since the order of elements in a set is irrelevant, we could obtain 23 other strings from the set c just by thinking of c's elements in a different order: these other strings will be equivalent to the first under an appropriate definition.

Before we can apply the techniques and theorems of formal languages, however, it will be necessary to represent codings as strings of symbols over a finite alphabet. We will want to discuss figures having an unbounded number of vertices, so we cannot provide distinct alphabet symbols for each of the vertices which we might encounter in coding an arbitrary figure. We shall do this by denoting the n^{th} vertex as "A" followed by n primes, which we will indicate in shorthand as A(n), or A'''... ''(n times), or occasionally, $A(')^n$. Thus we can use an alphabet having only three symbols, A, ', #. Using this system, a triangle could be coded

(*) A(1)A(2)#A(2)A(3)#A(3)A(1), that is,

(**) A'A''#A''A'''#A'''A'.

In general, a triangle would have the coding

(***) A(i)A(j)#A(j)A(k)#A(k)A(i), where i, j, and k are distinct positive integers, or some permutation of the line segments and vertices within this string (eight altogether).

---

[1]We use "#" instead of "," for line-delimitation in order to avoid confusing symbols which are used both formally and informally.

An alternative alphabet with only two symbols, 0 and 1, could equally well have been used, by representing A(n) by 01^n and # by 0. The above codings would become

(****) 0101100110111001110 1, and

$01^i01^j001^j01^k001^k01^i$.

A quick check will indicate that neither of these formulations is ambiguous, as it might seem at first. Another method for coding would be to use directly the symbols occurring in (*), namely the alphabet {A, #, (,), 0, 1, ..., 9}, where the vertices would now be distinguished by their indices in base ten, rather than by the number of primes.

In the following presentation we shall use the alphabet V = {A, #, '} for coding as in (**) above, because it provides a graphic representation which is easier to read than (****) and easier for a machine to scan than the alphabet using base ten indices, owing to the smaller number of symbols. Some results about different coding procedures will be given later.

Definition 3.1 $ls \in V^*$, where V = {A, #, '}, is a line segment[2] if and only if $ls = A(n_1)A(n_2) \ldots A(n_j)$ where $j \geq 2$, and each n_i is a positive integer with $i \neq i' \Rightarrow n_i \neq n_{i'}$. We will denote by Λ the set of all line segments.

This definition is clearly a translation of our previous notion of a line segment. Notice that the symbol "A" must be followed by one or more primes to be a vertex-label: this conforms to our earlier con-

[2]Again we use the term "line segment" rather than the more accurate term "coding for a line segment".

vention of using A(n) for the n^{th} vertex. For terminology, we will refer to a vertex A(n) in a line segment $ls \in \Lambda$ as an endpoint of ls if it happens that A(n) occurs as either the first or the last vertex in ls; otherwise A(n) is an interior vertex of ls.

Definition 3.2 $c \in V^*$ is a coding if and only if

1) $c = ls_1 \# ls_2 \# \ldots \# ls_n$ where each $ls_i \in \Lambda$ and $n > 0$,
2) no more than one vertex-label A(m) occurs simultaneously in ls_i and ls_j when $i \neq j$, and
3) if A(m) is an interior vertex-label in some line segment ls_i then A(m) also occurs as a vertex-label in some line segment ls_j where $i \neq j$.

We will denote the set of codings by C.

As before we let [ls] be the set of all vertex-labels which occur in the line segment ls; hence [A''A'A''''] = {A', A'', A''''}. We will denote by $[c]_v$ and $[c]_l$ the set of all vertex-labels and the set of all line segments respectively in a coding c[3].

Definition 3.3 Let a binary relation $\equiv$ be defined on C by $c_1 \equiv c_2$ if and only if there is a one-to-one function f mapping $[c_1]_v$ onto $[c_2]_v$ such that

1) if ls is a line segment in c_1 and $ls = A(n_1)A(n_2) \ldots A(n_j)$, then either $f(A(n_1))f(A(n_2)) \ldots f(A(n_j)) \in [c_2]_l$ or

[3]Put into the terminology of formal languages, we can describe $[c]_l$ as the set of all substrings of the coding c which are maximal with respect to the property of not containing the symbol "#", and a similar characterization applies to $[c]_v$.

$f(A(n_j))f(A(n_{j-1}))\ \ldots\ f(A(n_1))\ \varepsilon\ [c_2]_1$, and conversely,

2) if ls is a line segment in c_2 and $ls = A(n_1)A(n_2)\ \ldots\ A(n_j)$, then either $g(A(n_1))g(A(n_2))\ \ldots\ g(A(n_j))\varepsilon\ [c_1]_1$ or $g(A(n_j))g(A(n_{j-1}))\ \ldots\ g(A(n_1))\ \varepsilon\ [c_1]_1$, where $g = f^{-1}$.

If $c_1 \equiv c_2$ we call c_1 and c_2 equivalent codings.

Definition 3.4 A predicate P is a subset of V^*, where V is defined as above.

In the usual set-theoretic sense, predicates are identified as subsets of a specified set, but in the present formulation, it is possible to think of predicates as languages as well, in particular, as languages over the alphabet V. Hence certain predicates can be said to be regular, context-free, and so on. The predicates we have in mind will be geometric in nature: for example, the predicate K of connectedness would be defined $K = \{c\ \varepsilon\ V^* | c$ is a coding of a connected figure in the plane$\}$. Thus K is a sublanguage of $\mathcal{C}$. That K is well-defined is a consequence of the fact that $\mathcal{C}$ is well-defined and that "connectedness" is a generally recognised property applying to geometric figures.

An example of the type of results we wish to discuss is the following:

Theorem 3.5 The predicate P(x) given by "x is the coding for a line segment" is context-free and not regular.

Proof: The predicate P is given by $\{A(n)A(m) | n \neq m\}$. The following context-free grammar will generate these strings and no others:

$G = \langle V_N, V_T, P, S\rangle$, where $V_N = \{S, S_1, S_2, S_3\}$, $V_T = \{A, '\}$,

and where P is given by

1) $S \rightarrow AS_1$ 5) $S_2 \rightarrow$ 'A'

2) $S_1 \rightarrow 'S_1$ 6) $S \rightarrow AS_3$

3) $S_1 \rightarrow 'S_2$ 7) $S_3 \rightarrow S_3'$

4) $S_2 \rightarrow 'S_2'$ 8) $S_3 \rightarrow S_2'$

Rules 1-5 will generate strings of the form A(n)A(m) with $n > m \geq 1$ and rules 4-8 will generate similar strings with $m > n \geq 1$. The form of the rules shows that G is context-free.

This predicate cannot be regular because the memory limitations of a finite automaton fail to guarantee that n and m will be distinct integers for sufficiently large values of n and m.

This theorem illustrates a problem which we will deal with at some length, a problem which causes even the simplest imaginable predicate, short of the "empty" predicate, to fail to be regular. The difficulty lies in counting and comparing the number of primes occurring after an "A" in a coding. Simply put, a finite automaton cannot tell one vertex from another, unless a bound is placed on the vertex superscripts. This limitation is combinatorial rather than geometric, and we suggest one way of overcoming it:

Definition 3.6 An initial coding c is an element of C in which the only vertices appearing are A(1), A(2), ... A(n), where n is the number of distinct vertices present in the coded figure. If P is a predicate of codings, then P_I will be the predicate consisting of all codings in P which are initial. In particular, C_I is the set of all initial codings.

This definition reflects the natural way a person would assign vertex-labels to a figure, starting with A(1) rather than with A(1703). Using initial codings, a line segment would have only two distinct representations, A'A'' and A''A'. Notice that for any coding c there is an initial coding which is equivalent to it, and furthermore, that we can speak of initial codings, but not of initial strings in general. We will discuss initial and non-initial predicates in parallel until the two notions can be shown to converge.

We now prove some results about the predicates which are recognisable by finite automata.

Theorem 3.7 Let c be any coding and let $P_I(x)$ be the predicate given by "x is an initial coding which is equivalent to c". Then $P_I(x)$ is regular.

Proof: Immediate since there are only a finite number of initial codings which are equivalent to a given coding, and any finite language is regular.

As an example of such a predicate, consider the set of all initial codings for triangles. One such coding is A'A''#A''A'''#A'''A'. There are seven other ways to order the vertices within each line segment and six ways of ordering the lines, giving 48 distinct codings, each of which is initial and equivalent to the first.

It is easily proved that regular languages are closed under the set operations of union, intersection and complementation. As a result, initial predicates like the following are also regular:

1) x is not a triangle,

2) x is either a line or a quadrilateral,

3) x is neither a triangle nor a quadrilateral.

Note that the predicate $P_I(x)$ given by "x is not an inital coding equivalent to c" is satisfied not only by codings which are not inital and by codings which are initial but not equivalent to c, but also by strings in V^* which are not codings. Thus the complement of a language must be taken with respect to V^* and not C or C_I. In fact, it will be shown later that the predicate $P_I(x)$ mentioned above will not be regular in any case, for any coding c. This illustrates the result of taking complements with respect to C, instead of V^*.

We now examine the (non-initial) predicate T(x) given by "x is a triangle". In view of our previous results, we might expect T to be context-free; surprisingly, this is not the case.

Theorem 3.8 The predicate T(x) given by "x is a triangle" is context-sensitive and not context-free.

Proof: We will not give a proof of the second half of the theorem; similar languages are proved to be non-context-free in Ginsburg [p. 88 ff.]. However, the techniques required for this proof will be used later in this section. To show that T is context-sensitive, we give a complete grammar which will generate all and only codings for triangles. It should demonstrate the complexities involved in handling a simple, non-initial predicate. We use the notation:

$X \to A/B$ for $X \to A$ and $X \to B$,

$X(A,B) \to Y$ for $XA \to Y$ and $XB \to Y$, and

$X(A,B) \to (A,B)X$ for $XA \to AX$ and $XB \to BX$.

The production rules are as follows:

$S \rightarrow Q_1C_1B_1T$

$T \rightarrow L_1L_2L_3/L_1L_3L_2/L_2L_1L_3/L_2L_3L_1/L_3L_1L_2/L_3L_2L_1$

I $L_1 \rightarrow XY/YX$

$L_2 \rightarrow XZ/ZX$

$L_3 \rightarrow YZ/ZY$

$B_1 \rightarrow B_1B_1$

II $C_1 \rightarrow C_1C_1$

$Q_1 \rightarrow Q_1D_1$

$B_i(X,Y,',\#) \rightarrow (X,Y,',\#)B_i$ for i = 1, 2

III $B_1Z \rightarrow Z'B_2$

$B_2Z \rightarrow Z'$

$C_i(X,',\#) \rightarrow (X,',\#)C_i$ for i = 1, 2, 3, 4

IV $C_i(Y,Z) \rightarrow (Y',Z')C_{i+1}$ for i = 1, 2, 3

$C_4(Y,Z) \rightarrow (Y',Z')$

$D_i(',\#) \rightarrow (',\#)D_i$ for i = 1, 2, 3, 4, 5, 6

V $D_i(X,Y,Z) \rightarrow (X',Y',Z')D_{i+1}$ for i = 1, 2, 3, 4, 5

$D_6(X,Y,Z) \rightarrow (X',Y',Z')$

$Q_i(X,Y,Z) \rightarrow A'Q_{i+1}$ for i = 1, 2, 3, 4, 5

VI $Q_i(',\#) \rightarrow (',\#)Q_i$ for i = 1, 2, 3, 4, 5

$Q_6(X,Y,Z) \rightarrow A'$

(Total: 104 rules)

It can be seen from the rules that V_N = {S, T, L_1, L_2, L_3, B_1, B_2, C_1, C_2, C_3, C_4, D_1, D_2, D_3, D_4, D_5, D_6, Q_1, Q_2, Q_3, Q_4, Q_5, Q_6, X, Y, Z}. The rules in groups I and II are used to select one of the 48 possible orderings of line segments and vertices in the coding of a triangle and then to generate the proper number of primes for each vertex. The rules in groups III, IV and V shift these primes to the immediate

right of the appropriate vertex symbols. The last group of rules forces the complete application of all previous rules before generating the final terminal string.

Note that the preceding grammar can be generalized to produce all codings of n-gons for a fixed value of n, and hence this predicate is also context-sensitive.

Theorem 3.9 C and C_I are context-sensitive languages.

Proof: The suggestion to use LBA's in this proof was originally due to R. Roskies. The plan is to enter a string of symbols from the alphabet V = {A, #, '} as input to a suitably defined LBA. To ensure that the tape head does not leave that portion of the tape which contains the input string, it is customary to surround the input with end-markers, ¢ on the left and $ on the right. The LBA will proceed through a series of tests on the input, each of which will leave the input string unchanged upon completion. The tests we have in mind are these:

1) check that each maximal substring of the input containing no #'s satisfies the definition of a line segment,
2) check that no pair of such line segments has more than one vertex in common,
3) check that each vertex occurs either as an endpoint of a line segment or at least twice in the coding.

In the case of C_I we make an additional test:

4) check that the only vertices which occur are A(1), A(2), ... A(n) for some n.

We now give an explicit program which performs the first task. This will illustrate the technique involved and indicate that an LBA is adequate to perform such tasks.

Routine R1

The transition function is:

q_o	A	q_1	A	R	R1 checks that the input, that is,
q_1	'	q_2	'	R	the tape between the ¢ and the $,
q_2	'	q_2	'	R	is in the proper form: that it
q_2	A	q_3	A	R	starts with an A, ends with a ',
q_3	'	q_4	'	R	has at least one ' after each A,
q_4	'	q_4	'	R	has at least two A's between each
q_4	A	q_3	A	R	pair of #'s and between the ¢ and
q_4	#	q_o	#	R	the first # and also between the last
q_4	$	q_5	$	L	# and the $, and that each # is
q_5	α	q_5	α	L	followed by an A. The tape head will
q_5	¢	q_6	¢	R	start and end at the leftmost square

where α stands for any symbol except ¢. The start state and the final state are q_o and q_6, respectively. of the input.

Routine R2

The transition function is:

q_o	A	q_1	A	R	R2 starts at the leftmost A in the
q_1	'	q_1	'	R	first line segment and checks that
q_1	A	q_2	A	L	no other vertex in the same line
q_2	'	q_3	X	R	segment has the same number of '

q_3	A	q_4	A	R
q_4	'	q_5	Y	L
q_5	Y	q_5	Y	L
q_5	A	q_5	A	L
q_5	'	q_5	'	L
q_5	X	q_6	X	L
q_6	X	q_6	X	L
q_6	'	q_7	X	R
q_7	X	q_7	X	R
q_7	A	q_7	A	R
q_7	'	q_7	'	R
q_7	Y	q_8	Y	R
q_8	Y	q_8	Y	R
q_8	'	q_5	Y	L

---

q_6	A	q_9	A	R
q_9	X	q_9	X	R
q_9	A	q_{10}	A	R
q_{10}	'	q_{10}	'	R
q_{10}	A	q_{10}	A	R
q_{10}	Y	q_{11}	Y	R
q_{11}	Y	q_{11}	Y	R
q_{11}	'	q_{12}	'	R
q_{12}	'	q_{12}	'	R
q_{12}	A	q_{13}	A	R

---

q_8	A	q_{13}	A	R

---

symbols following it. If such is the case, the second vertex is similarly tested, and so on. When every vertex in the first line segment has been tested, the LBA proceeds to the second and succeeding line segments.

---

q_{13}	'	q_{14}	Y	L
q_{14}	'	q_{14}	'	L
q_{14}	A	q_{14}	A	L
q_{14}	Y	q_{15}	'	L
q_{14}	X	q_{15}	X	L
q_{15}	X	q_{15}	'	L
q_{15}	A	q_{16}	A	R
q_{16}	A	q_{16}	A	R
q_{16}	'	q_{16}	'	R
q_{16}	X	q_{16}	X	R
q_{16}	Y	q_5	Y	L

---

q_{12}	#	q_{17}	#	L
q_{12}	$	q_{17}	$	L
q_{17}	A	q_{17}	A	L
q_{17}	'	q_{17}	'	L
q_{17}	Y	q_{17}	'	L
q_{17}	X	q_{18}	'	L
q_{18}	X	q_{18}	'	L
q_{18}	'	q_{19}	'	R

q_{18}	A	q_{19}	A	R
q_{19}	'	q_{19}	'	R
q_{19}	A	q_1	A	R
q_8	\$	q_{17}	\$	L
q_8	#	q_{17}	#	L
q_1	#	q_o	#	R
q_1	\$	q_{20}	\$	L
q_{20}	A	q_{20}	A	L
q_{20}	'	q_{20}	'	L
q_{20}	#	q_{20}	#	L
q_{20}	¢	q_{21}	¢	R

The start state and the final state are q_o and q_{21}, respectively.

The second test uses the techniques employed in R2 above: the automaton proceeds through each pair of line segments comparing all vertices, one from each line segment, and halting in a non-final state if a pair of line segments have more than one vertex in common.

The third test is almost identical with the second, except that it is necessary to check for duplication only those vertices which do not occur as endpoints, and they must be checked against all vertices which occur.

The fourth test breaks into two parts: first, the longest string of primes is located and replaced with X's. Then the input is checked for a string of primes which is exactly one symbol shorter than the

string of X's. If one is found, we replace one X with a ' and repeat the test until only one X remains. The LBA then replaces this X with a ' and halts in a final state.

The following theorem (3.11) will be of fundamental importance when discussing the limitations of formal languages in geometric applications. However, we first state a result which provides a partial characterization of context-free languages, due originally to Bar-Hillel, Perles and Shamir. It is referred to variously as the "uvwxy" theorem in Hopcroft and Ullman [p. 57] and the "xuwvy" lemma in Ginsburg [p. 84].[4]

Lemma 3.10 Let $\mathcal{L}$ be a context-free language. Then there exist integers p and q with the property that any $z \in \mathcal{L}$ with $|z| > p$ can be written as $z = uvwxy$ where $|vwx| \leq q$, either $v \neq e$ or $x \neq e$, and where $z_i = uv^iwx^iy \in \mathcal{L}$ for each integer $i \geq 0$.

Theorem 3.11 ("Star" Theorem) For $n > 1$, let ST(n) be the coding for an (n-1)-pointed "star" figure, A(1)A(2)#A(1)A(3)#.....#A(1)A(n). Let P be any predicate of codings which contains ST(n) for infinitely many n. Then P cannot be context-free.

Proof: Let P be any such predicate of codings, that is, $P \subseteq C$ and $ST(n) \in P$ for infinitely many n, and suppose that P is context-free. We can clearly select an $n > 0$ and a $z = ST(n) \in P$ such that $|z| > p$, where p is determined by P as in the lemma. Hence $z = uvwxy$. We shall

[4]The "uvwxy" theorem provides a necessary condition for context-free languages, but it is not known whether the condition is also sufficient.

examine v (or x if v = e).

First, since v can be assumed to be non-empty, suppose v contains an occurrence of the symbol #, that is, v = α#β, where α and β ε V^* (and possibly one or both is empty). Then v^i = α#βα#βα# ... #β = α(#βα)$^{i-1}$#β. If α and β are both empty, then z_i = uv^iwx^iy will contain a repetition of the # symbol i times, and if either α or β is non-empty, z_i will contain i-1 repetitions of #βα. Either way, even if βα should be a well-formed line segment, z_i will not be a well-formed coding, regardless of what wxy looks like. Thus v cannot contain #.

Now suppose that v contains the symbol A. If so, then v is A'A(')^j, 'A(')^j, or A(')^j for some integer j ≥ 0. Hence v^i will be A'A(j)A'A(j) ... A'A(j), 'A(j+1)$^{i-1}$A(j), or A(j)^i. None of these substrings can occur in a coding, since each represents a repetition of vertices within a line segment. Thus v cannot contain the symbol A.

Hence v must be a string of ' symbols. Suppose v consists of all the ' symbols following some occurrence of an A in z. Unless wxy = e, a possibility we can avoid by taking n > q+1, then v is bounded on the left by an A and on the right by either an A or a #. By lemma 3.10, z_0 = uwy ε P. But uwy will have an occurrence of either an AA or an A#, neither of which can occur in a proper coding.

Therefore, v is a string of k ≥ 1 ' symbols, which occurs in a line segment A'A(n) in z, where n > k. In z_0 = uwy, this line segment becomes A'A(n-k) if n-k > 1, or it becomes A'A': in the first case, it duplicates a line segment occurring to the left of A'A(n), in the second it becomes ill-formed. The occurrence of x, if x ≠ e, will not affect the argument, since it is to the right of v. Thus we see that z cannot be represented in such a way that z_i ε P for each i ≥ 0; hence P is not context-free.

Corollary 3.12 Neither C nor C_I is context-free.

Proof: Clearly ST(n) $\in C_I \subseteq C$ for all integers n > 1.

Theorem 3.9 and Corollary 3.12 characterize the two methods of coding geometric figures. Using these results we can now demonstrate the convergence of these two methods at the context-sensitive level.

Corollary 3.13 Part I Let P be any predicate of codings, that is, $P \subseteq C$, which is context-sensitive. Then the corresponding initial predicate P_I is also context-sensitive.

Part II Conversely, let P_I be any predicate of initial codings, that is, $P_I \subseteq C_I$, which is context-sensitive and which is invariant under coding equivalence: if $x \in C_I$ and $y \in C_I$ and $x \equiv y$, then $P_I(x)$ if and only if $P_I(y)$. Then the extension of P_I to non-initial codings, defined by P(x) if and only if for some y, $x \equiv y$ and $P_I(y)$, will also be context-sensitive.

Proof, Part I: To convert from P to P_I we need only add an additional test to the routines used in the testing of P, namely a verification that the input is initial, which in Theorem 3.9 was seen to be context-sensitive.

Part II: Let P and P_I satisfy the hypotheses of Part II of the corollary. To determine if P(x) holds, we first check that x is a coding, then we transform x into an equivalent initial coding y and check whether P(y) holds. The first and last of these tests are known already to be within the scope of some LBA, so we need only show that an LBA can transform a given coding into an equivalent initial one. Note that we can always find an equivalent initial coding whose length is not greater

than that of the original coding; it is not true that an initial coding is always no longer than a non-initial equivalent coding.

We can define the LBA which converts a coding to an initial one as follows: we first locate the shortest string of ' symbols in the input and replace it with a single temporary prime-marker M followed by enough N symbols to fill out the string. Then the rest of the input is scanned for strings of ' symbols having the same length, which are treated similarly. Then the next longest strings of ' symbols are located and replaced by two M's followed by N's, and so on. When the process is complete, the N's will be eliminated and the M's will become ' symbols again. The effect is to convert the vertex of lowest index into an A(1), the vertex of next lowest index into an A(2), and so on. It is clear that such a process can be carried out within the given amount of tape.

An interesting question is whether the preceding corollary holds when "context-sensitive" is replaced throughout by "context-free" or "regular". Obviously Part II will fail under either replacement. The predicate $T_I(x)$ given by "x is an initial coding for a triangle" is regular (Theorem 3.7) and hence also context-free, but T(x), the extension of $T_I(x)$ to non-initial codings, is neither regular nor context-free (Theorem 3.8).

Part I of the Corollary is problematic in that we have virtually no examples of non-initial predicates of codings which are regular and only one which is context-free (Theorem 3.5). For this one example, Part I certainly holds, but for now we will leave the question open, for the reason that an answer to it will not be likely to have any application

to the kind of predicates we are discussing here.

We conclude this section with some results about previously-mentioned predicates.

Theorem 3.14 The predicate K(x) given by "x is the coding of a connected figure" is context-sensitive and not context-free.

Proof: The simple and obvious technique for handling connectedness with an LBA is this: we first check that the input is a proper coding and then mark with a new symbol X each of the vertices in the first line segment. Then we iterate two routines until no change results: the first is to mark with an X each additional occurrence of vertices already so marked, and the second is to mark with X's each vertex occurring in a line segment which contains an X-marked vertex. This will identify those vertices which are connected to the first vertex. If all vertices are marked at the end of the process, then the figure is connected.

That K is not context-free is a consequence of the "Star" Theorem.

Theorem 3.15 Let $c \in \mathcal{C}$ and let P(x) be the predicate "$x \in \mathcal{C}$ and $x \neq c$". Then both P and P_I are context-sensitive and not context-free.

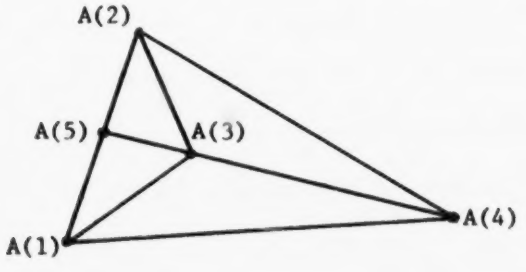
Proof: Neither P nor P_I is context-free, for the "Star" Theorem applies to each, even if $c = ST(n)$ for some $n > 1$.

An LBA to recognise elements of P or P_I would first determine whether or not the input is a coding and in the case of P_I whether or not it is initial. Since there are a finite number of initial codings equivalent to c, the LBA would then transform its input to an initial coding if it is not already and compare the result with each of the possible initial codings.

In view of Theorem 3.15, we might ask whether the complement of any context-sensitive predicate remains context-sensitive when taken with respect to $\mathcal{C}$. The answer is that we simply do not know at this point, for this question seems to be a special case of the larger problem of whether the class of context-sensitive languages is closed under complementation. An affirmative answer to this will imply that the class of languages accepted by deterministic LBA's is the same as that accepted by nondeterministic LBA's.

One last predicate which was mentioned earlier is the "figure in context" predicate, an example being "x contains a triangle". We can tell if a given figure contains another as a subfigure by a correct succession of vertex deletions, removing the lines connecting each vertex at the same time. On the level of codings, the process would work like this: if c is a coding and A(n) is a vertex in c, we can obtain a new coding c' from c by first deleting every occurrence of A(n) in c, then deleting any vertex A(m) which had occurred as the only other vertex on a line segment A(n)A(m) or A(m)A(n) in the original coding c, but deleting only those occurrences of A(m) which are as just described. Last, each time we delete a vertex A(m) we must also remove one of the # symbols which had been adjacent to it. For example, suppose we wish to remove vertex A(3) in the following figure:

Figure 1



A coding for this figure would be:

A(1)A(5)A(2)#A(5)A(3)A(4)#A(1)A(3)#A(2)A(3)#A(1)A(4)#A(2)A(4).

To delete vertex A(3) we first remove each occurrence of A(3):

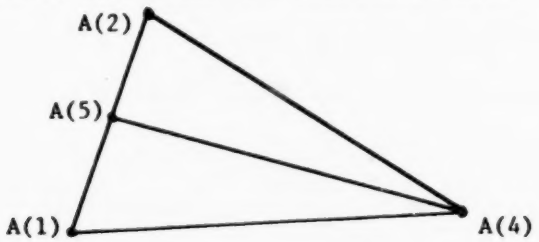
A(1)A(5)A(2)#A(5)A(4)#A(1)#A(2)#A(1)A(4)#A(2)A(4),

then we delete the occurrences of A(1) and A(2) in the third and fourth line segments respectively, together with one of the # symbols for each:

A(1)A(5)A(2)#A(5)A(4)#A(1)A(4)#A(2)A(4).

The resulting coding represents the following figure:

Figure 2



Thus if F is a figure, we can recognise F within the context of another figure by some succession of such deletions.

Theorem 3.16 The predicate $P_F(x)$ given by "x contains F as a subfigure" is context-sensitive.

Proof: We can design an LBA which takes an input string, checks to determine if it represents a coding, and then converts the coding symbol by symbol to ordered pairs: A becomes (A,A), ' becomes (',') and # becomes (#,#). Then the LBA tries successive deletions of vertices writing the result in the second coordinate of these ordered pairs, and replacing each deleted symbol by a null symbol N. After each deletion, a new coding would be initialized and compared with each of the finitely many initial codings for F. When a particular succession of deletions fails to find a coding for F, the original coding is rewritten into the second coordinates, and the process starts again. The LBA halts

in a final state only when a coding for F is found.

Corollary 3.17 The predicate "x is a TNP coding" is context-sensitive.

Proof: We use Kuratowski's theorem and the preceding result to locate one of the two "forbidden" subcodings within the given coding. If one or both is found, the coding is TNP.

The foregoing results amply suggest that context-sensitive languages play the central role in characterizing those types of geometric properties under discussion here. Even with the restricted amount of "geometry" available to us, the full power of context-sensitivity was necessary to handle all but the simplest figure-classes, and we feel that there is every reason to believe that similar investigations will reach much the same conclusions.

The disturbing aspect of this study is that linguistics as it is applied to the study of natural languages, centers upon the context-free property; in fact, Ginsburg refers to languages which are not context-free as "non-languages". Thus much linguistic analysis cannot be extended to include geometry, even in the simple formulation presented here. Among the implications of this fact we can mention the non-existence of derivation trees for codings and the impossibility of left-to-right generation of terminal symbols in codings, both of which are used in the analysis of natural languages.

SECTION 4: INVARIANCE THEOREMS

Many of the results in the preceding section seem to depend heavily upon the choice of coding procedure employed: it is natural to ask whether some other method of coding figures would have given more optimistic results. The theorems in this section will help to generalize the results in Section 3 to a larger class of coding procedures.

Definition 4.1 A finite automaton with output (FAO) is a quintuple $\mathcal{F} = \langle K, \Sigma, \Lambda, q_o, \delta \rangle$ in which

1) K is a finite, non-empty set (the set of states),
2) Σ and Λ are finite non-empty sets (the input and output alphabets, respectively),
3) $q_o \in K$ (the start state), and
4) $\delta: K \times \Sigma \to K \times \Lambda$ (the transition function).

An FAO operates like a finite automaton except that instead of accepting or rejecting an input string, it generates a new string of symbols, of length equal to the length of the input string. Thus we can think of an FAO as a function from Σ^* to Λ^*. We will define a non-deterministic FAO analogously, except that we now allow $\delta: K \times \Sigma \to \mathcal{P}(K \times \Sigma) - \{\phi\}$, the set of non-empty subsets of $K \times \Sigma$. When $\mathcal{F}$ is in a state q and scanning a symbol $v \in \Sigma$, the range of possible moves for $\mathcal{F}$ is any $(q',w) \in \delta(q,v)$. If α is an input string, we will let $\mathcal{F}(\alpha)$ denote the set of all output strings which can be generated from α by the NDFAO $\mathcal{F}$. We shall think of deterministic FAO's as special cases

of NDFAO's, so that many theorems about the latter will apply a fortiori to the former.

As an example of an FAO, consider $\mathcal{F} = \langle\{q_o\}, \{A, ', \#\}, \{0, 1\}, q_o, \delta\rangle$, where δ is defined:

$$\delta(q_o,A) = (q_o,0)$$

$$\delta(q_o,') = (q_o,1)$$

$$\delta(q_o,\#) = (q_o,0)$$

If the string A'A''#A''A'''#A'''A' is used as input for $\mathcal{F}$, the output is 01011001101110011101. The effect is to transform a coding in the sense used in Section 3 into one of the similar codings mentioned on page 20.

Definition 4.2 Let $\mathcal{L}$ and $\mathcal{L}'$ be languages over the alphabets W and W' respectively. Then $\mathcal{L}'$ is a regular transform of $\mathcal{L}$ in case there is a NDFAO $\mathcal{F}$ such that $\mathcal{L}'$ is the image of $\mathcal{L}$ under $\mathcal{F}$, denoted $\mathcal{L}' = \mathcal{F}(\mathcal{L})$. When $\mathcal{L}$ is a specified language and $\mathcal{F}$ is a NDFAO whose input alphabet includes the alphabet of $\mathcal{L}$, we will call $\mathcal{F}(\mathcal{L})$ the regular transform of $\mathcal{L}$ under $\mathcal{F}$.

We have defined a regular transform with respect to the most general type of FAO, namely, non-deterministic, and this much generality may not always be necessary. Clearly the case which is most interesting is when the FAO relating two languages is not merely deterministic, but one-to-one as well. The reason for the definition as it stands, is that it will be useful to be able to define the inverse of an FAO which would not in general be an FAO unless the original FAO were supposed to be one-to-one. Note that even when NDFAO's are employed, the image of a finite language

is finite and the image of an infinite language is infinite.

Remark: If $\mathcal{L}'$ is a regular transform of $\mathcal{L}$, $\mathcal{L}$ may not be a regular transform of $\mathcal{L}'$, even if the FAO is deterministic and one-to-one on $\mathcal{L}$. For example, let $\mathcal{L} = \{a^n b^n | n > 0\}$. $\mathcal{L}$ is context-free and not regular. Let $\mathcal{F} = \langle\{q_o\}, \{a, b\}, \{1\}, q_o, \delta\rangle$, where δ is given by $\delta(q_o,a) = (q_o,1)$ and $\delta(q_o,b) = (q_o,1)$. Then $\mathcal{F}(\mathcal{L}) = \mathcal{L}' = \{1^n | n$ is an even positive integer$\}$, and $\mathcal{L}'$ is regular. By the following theorem, $\mathcal{L}$ cannot be a regular transform of $\mathcal{L}'$.

Theorem 4.3 If $\mathcal{L}$ is regular and $\mathcal{L}'$ is a regular transform of $\mathcal{L}$, then $\mathcal{L}'$ is regular.

Proof: Let $G = \langle V_N, V_T, P, S\rangle$ be a regular grammar for $\mathcal{L}$ and let the NDFAO be $\mathcal{F} = \langle K, V_T, W, q_o, \delta\rangle$ where $\mathcal{L}' \subseteq W^*$ and $\mathcal{F}(\mathcal{L}) = \mathcal{L}'$. Define a new grammar $G' = \langle V_N \times K, W, P', (S,q_o)\rangle$ where the rules in P' are specified as follows:

If $A \to vB$ is a rule in P, then for every $q \in K$, P' will contain all the rules $(A,q) \to w(B,q')$ where $(q',w) \in \delta(q,v)$.

If $A \to v$ is a rule in P, then for every $q \in K$, P' will contain the rules $(A,q) \to w$ where for some $q' \in K$, $(q',w) \in \delta(q,v)$.

Now we prove that $\mathcal{L}' = L(G')$. Let $\alpha \in \mathcal{L}'$ where $\alpha = w_1 w_2 \ldots w_n$. Since $\alpha \in \mathcal{L}'$, there is a $\beta \in \mathcal{L}$, $\beta = v_1 v_2 \ldots v_n$, such that $\alpha \in \mathcal{F}(\beta)$. G generates $\mathcal{L}$, so we have a sequence of rules in P: $S \to v_1 S_1$, $S_1 \to v_2 S_2$, $\ldots S_{n-1} \to v_n$. Also since $\alpha \in \mathcal{F}(\beta)$, we have in K a sequence of states $q_o, \ldots q_n$ such that $(q_o,v_1) \to (q_1,w_1)$, $(q_1,v_2) \to (q_2,w_2)$, $\ldots$ $(q_{n-1},v_n) \to (q_n,w_n)$, that is, $(q_i,w_i) \in \delta(q_{i-1},v_i)$ for $i = 1, 2, \ldots n$. Hence in P' we have a sequence of rules $(S,q_o) \to w_1(S_1,q_1)$, $(S_1,q_1) \to w_2(S_2,q_2)$,

... $(S_{n-1},q_{n-1}) \rightarrow w_n$. Thus $\alpha = w_1w_2 \ldots w_n \in L(G')$.

Conversely, let $\alpha \in L(G')$, $\alpha = w_1w_2 \ldots w_n$. Then there is in P' a sequence of rules $(S,q_o) \rightarrow w_1(S_1,q_1)$, $(S_1,q_1) \rightarrow w_2(S_2,q_2)$, ... $(S_{n-1},q_{n-1}) \rightarrow w_n$. Hence we have for some sequence of v_i's in V_T that $(q_1,w_1) \in \delta(q_o,v_1)$, $(q_2,w_2) \rightarrow \delta(q_1,v_2)$, ... , $(q_n,w_n) \rightarrow \delta(q_{n-1},v_n)$. Hence there is in P a sequence for rules $S \rightarrow v_1S_1$, $S_1 \rightarrow v_2S_2$, ... $S_{n-1} \rightarrow v_n$. Thus $\beta = v_1v_2 \ldots v_n \in \mathcal{L}$, and $\alpha \in \mathcal{F}(\beta)$. So $\alpha \in \mathcal{L}'$.

Thus we have proved that $\mathcal{L}' \subseteq L(G')$ and $L(G') \subseteq \mathcal{L}'$, so the two are equal.

Theorem 4.4 If $\mathcal{L}$ is context-free and $\mathcal{L}'$ is a regular transform of $\mathcal{L}$, then $\mathcal{L}'$ is context-free.

Proof: Let $\mathcal{L} \subseteq V^*$, let $G = \langle V_N, V, P, S\rangle$ be a context-free grammar for $\mathcal{L}$, and let $\mathcal{F} = \langle K, V, W, q_o, \delta\rangle$ be a NDFAO mapping $\mathcal{L}$ to $\mathcal{L}'$, where $\mathcal{L}' \subseteq W^*$. We may assume that the rules in P are in Chomsky normal form, that is, each rule is of the form $A \rightarrow BC$ or $A \rightarrow v$, where $A, B, C \in V_N$ and $v \in V$. For each $A \in V_N$ and $q \in K$ define $\mathcal{A}(A,q) = \{q' \in K \mid A \rightarrow_G^* \alpha$ for some $\alpha \in V^*$ and $(q,w) \in \delta^*(q',\alpha)$ for some $w \in W\}$, where by δ^* we mean the natural extension of δ to strings [see p. 15]. Intuitively $\mathcal{A}(A,q)$ is the set of all states in K which are in a sense "final" states when δ is applied to a string α with q as start state, where α is some terminal string generated from the variable A. Thus $\mathcal{A}(S,q_o)$ would be the set of "final" states for strings in $\mathcal{L}$. Notice that if A is a variable in V_N which occurs in at least one derivation in G then $A \rightarrow_G^* \alpha$ for some $\alpha \in V^*$, and hence $\mathcal{A}(A,q)$ is non-empty for such variables.

Define a grammar $G' = \langle V_N \times K, W, P', (S,q_o)\rangle$ where the rules in P' are specified as follows:

1) If $A \to BC$ is in P then we will include in P' each rule of the form $(A,q) \to (B,q)(C,q')$ where q is any element of K and $q' \in \mathcal{Q}(B,q)$.

2) If $A \to v$ is in P then we will include in P' all rules of the form $(A,q) \to w$ where for some $q' \in K$, $(q',w) \in \delta(q,v)$.

Notice that G' is a context-free grammar.

Part I: Let $\beta = w_1w_2 \ldots w_n \in \mathcal{L}'$. There is an $\alpha = v_1v_2 \ldots v_n \in \mathcal{L}$ with $\beta \in \mathcal{F}(\alpha)$ and hence in K we have elements $q_o, q_1, \ldots q_n$ such that $(q_1,w_1) \in \delta(q_o,v_1)$, $(q_2,w_2) \in \delta(q_1,v_2)$, $\ldots$ $(q_n,w_n) \in \delta(q_{n-1},v_n)$. Let D be a derivation of α in G, arranged so that all rules of the form $A \to BC$ precede rules of the form $A \to v$. Thus $S \to_G^* A_1A_2 \ldots A_n$ for some sequence of variables in V_N, such that $A_i \to v_i$ is in P for $i = 1, 2, \ldots n$. We will establish that $(S,q_o) \to^* (A_1,q_o)(A_2,q_1) \ldots (A_n,q_{n-1})$ in G'.

Certainly for some sequence $r_o, r_1, \ldots r_n$ in K we have $(S,q_o) \to_{G'}^* (A_1,r_o)(A_2,r_1)\ldots(A_n,r_{n-1})$, since each rule $A \to BC$ used in the derivation of $A_1A_2 \ldots A_n$ has counterparts in P' of the form $(A,q) \to (B,q)(C,q')$ for all $q \in K$ and for $q' \in \mathcal{Q}(B,q)$, since $\mathcal{Q}(B,q)$ is non-empty. Thus the derivation $S \to_G^* A_1A_2 \ldots A_n$ can be duplicated in G'. We will use induction to prove that it is possible for $r_o = q_o$, $r_1 = q_1$, $\ldots$ $r_{n-1} = q_{n-1}$.

First note that $r_o = q_o$ as a result of the construction of the rules: the second component of the leftmost variable in any derivation in G' of a non-terminal string from (S,q_o) must be q_o.

Now suppose the result to be true for i-1, that is, that $(S,q_o) \to^* (A_1,q_o)(A_2,q_1) \ldots (A_i,q_{i-1})(A_{i+1},r_i) \ldots (A_n,r_{n-1})$ in G'. In the

derivation tree of α, locate the least upper bound X of the variables A_i and A_{i+1}. Then for variables Y and Z in V_N we have that $X \to YZ$ is in P; that either $Y = A_i$ or for some j $(1 \leq j < i)$ $Y \to^* A_jA_{j+1} \ldots A_i$; and that either $Z = A_{i+1}$ or for some k $(i+1 < k \leq n)$ $Z \to^* A_{i+1}A_{i+2} \ldots A_k$. In P' we have the rules $(X,q) \to (Y,q)(Z,q')$ for each $q \in K$ and $q' \in \mathcal{A}(Y,q)$, and in particular $(X,q_{j-1}) \to (Y,q_{j-1})(Z,q')$ where $q' \in \mathcal{A}(Y,q_{j-1})$. Furthermore notice that since $Y \to^* v_jv_{j+1} \ldots v_i$ in G and $(q_i,w_i) \in \delta^*(q_{j-1},v_jv_{j+1} \ldots v_i)$, we must have $q_i \in \mathcal{A}(Y,q_{j-1})$, and therefore $(X,q_{j-1}) \to (Y,q_{j-1})(Z,q_i)$ is in P'.

Notice now that by induction (X,q_{j-1}) must occur as a variable in the derivation of $(A_1,q_o)(A_2,q_1) \ldots (A_i,q_{i-1})(A_{i+1},r_i) \ldots (A_n,r_{n-1})$ and hence it is possible for $r_i = q_i$. Thus the induction step is complete and $(S,q_o) \to^* (A_1,q_o)(A_2,q_1) \ldots (A_n,q_{n-1})$.

It remains to show that $(A_i,q_{i-1}) \to w_i$ is in P' for i = 1, 2, ... n, but this follows from the fact that $A_i \to v_i$ is in P and $(q_i,w_i) \in \delta(q_{i-1},v_i)$. Thus $\beta \in L(G')$.

Part II: Now let $\beta = w_1w_2 \ldots w_n \in L(G')$. In G' we have a derivation $(S,q_o) \to^* (A_1,q_o)(A_2,q_1) \ldots (A_n,q_{n-1})$ and rules $(A_i,q_{i-1}) \to w_i$ for i = 1, 2, ... n. Each rule in P' of the form $(A,q) \to (B,q)(C,q')$ corresponds to a unique rule in P, $A \to BC$, so in G we have $S \to^* A_1A_2 \ldots A_n$. In addition the rules $(A_i,q_{i-1}) \to w_i$ correspond to rules $A_i \to v_i$ in P, where each v_i is such that $(q_i,w_i) \in \delta(q_{i-1},v_i)$. Thus we have $S \to^* \alpha = v_1v_2 \ldots v_n$ in G. So $\alpha \in \mathcal{L}$, and $\beta \in \mathcal{F}(\alpha)$ since $(q_1,w_1) \in \delta(q_o,v_1)$, $(q_2,w_2) \in \delta(q_1,v_2)$, ... $(q_n,w_n) \in \delta(q_{n-1},v_n)$. So $\beta \in \mathcal{L}'$.

Parts I and II show that L(G') both contains and is contained in $\mathcal{L}'$, hence $L(G') = \mathcal{L}'$ and $\mathcal{L}'$ is context-free.

Theorem 4.5 Let $\mathcal{L}$ be context-sensitive and $\mathcal{L}'$ be a regular transform of $\mathcal{L}$. Then $\mathcal{L}'$ is context-sensitive.

Proof: Let $\mathcal{F} = \langle K, V, W, q_0, \delta \rangle$ be a NDFAO which maps $\mathcal{L}$ onto $\mathcal{L}'$, where $\mathcal{L} \subseteq V^*$ and $\mathcal{L}' \subseteq W^*$. We define a special three-track LBA T to recognise words in $\mathcal{L}'$. T will accept a word $\beta = w_1 w_2 \ldots w_n \in W^*$ as input and change each symbol $w_i \in \beta$ to a triple $(w_i, 0, 0)$. If $\beta \in \mathcal{L}'$ then there is an $\alpha = v_1 v_2 \ldots v_n$ such that $\beta \in \mathcal{F}(\alpha)$: T will find α as follows: first T generates nondeterministically a string $\alpha \in V^*$ with $|\alpha| = n$, and stores α symbol by symbol in the blank second coordinates of the input. Then T applies δ to α nondeterministically, generating elements of $\mathcal{F}(\alpha)$ and storing each in turn in the third coordinate space of the input. When an element of $\mathcal{F}(\alpha)$ is found which equals β, T will then test α to determine whether $\alpha \in \mathcal{L}$, using the fact that $\mathcal{L}$ is context-sensitive. If not, T will discard α and start over with another $\alpha' \in V^*$ with $|\alpha'| = n$. T will halt in a final state when an element λ of $\mathcal{L}$ is found with $\beta \in \mathcal{F}(\lambda)$. At each point in the test to determine whether a particular $\alpha \in V^*$ belongs to $\mathcal{L}$, T can nondeterministically exit and begin computation over again with a (possibly) different value for α. This will prevent T from becoming bogged down in computations for an incorrect value of α.

Corollary 4.6 If $\mathcal{L}$ is Type 0 and $\mathcal{L}'$ is a regular transform of $\mathcal{L}$, then $\mathcal{L}'$ is also Type 0.

Proof: The technique employed in Theorem 4.5 generalizes to Turing machines, which may need more computing space to determine whether $\alpha \in \mathcal{L}$, but otherwise work the same.

One major limitation of regular transforms would seem to be the fact that the length of the output string must equal that of the input string. There are tricks however which can help overcome this problem in some special cases. Suppose our coding procedure had been this: the n^{th} vertex is coded by 1^n; 0 is a vertex-delimitation symbol; # is a line-delimitation symbol. The triangle

* A(1)A(3)#A(4)A(1)#A(3)A(4)

would be coded

** 10111#111101#11101111.

The number of symbols in * is 24, the number in ** is 21. If we modify the second coding procedure slightly, we can make the number of symbols in a "new" coding correspond to the number of symbols in the original. Suppose we consider the "new" coding to be over the alphabet {0, 1, #, S}, where S is a new symbol, and the procedure is modified so that each coding starts with an S and the lines are now delimited with ## instead of # as before:

*** S10111##111101##11101111.

It is easily seen that this procedure results in codings which have the same length as codings in *, relative to the choice of vertex-labels. If the change is not objectionable, we now have a coding procedure which is a regular transform of our original one. The FAO which accomplishes this is given by:

$\mathcal{F} = \langle \{q_o, q_1, q_2\}, \{A, ', \#\}, \{S, 0, 1, \#\}, q_o, \delta \rangle$ where

δ is defined by

$\delta(A,q_o) = (S,q_1)$ $\quad$ $\delta(\#,q_1) = (\#,q_2)$

$\delta('',q_1) = (1,q_1)$ $\quad$ $\delta(A,q_2) = (\#,q_1)$

$\delta(A,q_1) = (0,q_1)$

We can use similar tricks to "shrink" or "stretch" the output string, but a better way is to generalize our definition of regular transform to include the case where a NDFAO can output a finite number of symbols and possibly the empty symbol at each step, instead of exactly one symbol. The appropriate definitions are these:

Definition 4.7 A generalized NDFAO (GFAO)[5] is a quintuple $\mathcal{F} = \langle K, \Sigma, \Lambda, q_0, \delta \rangle$ where

1) K, Σ, and Λ are finite non-empty sets,

2) $q_0 \in K$,

3) δ is a function from $K \times \Sigma$ to non-empty finite subsets of $K \times \Sigma^*$.

Definition 4.8 If $\mathcal{L} \subseteq V^*$ and $\mathcal{L}' = \mathcal{F}(\mathcal{L})$ for some GFAO $\mathcal{F}$, then $\mathcal{L}'$ is called a generalized regular transform (GRT) of $\mathcal{L}$.

Theorem 4.9 The GRT of a regular (context-free) language is regular (context-free).

Proof: Let $\mathcal{L}$ be regular and let G be a regular grammar for $\mathcal{L}$. We can find a grammar G' for $\mathcal{L}'$ as we did in the proof of Theorem 4.3: the rules in P' will now be of the form $A \to \alpha B$ and $A \to \alpha$, where A and B are variables of G' and $\alpha \in W^*$, possibly $\alpha = e$. To show that $\mathcal{L}'$ is

[5]These definitions and the following theorems were discovered to be minor variations of what is known in the literature as a generalized sequential machine and similar results for so-called "gsm mappings" can be found in Hopcroft and Ullman [pp. 128-130]. However, the results given here were developed independently as extensions of the notion of a regular transform and the proofs in this paper are in no way related to the proofs in Hopcroft and Ullman. We include these results for completeness.

regular we will replace the rules in P' with new equivalent ones.

First, if $A \to \alpha B$ is in P' for some $\alpha = v_1 v_2 \ldots v_n$ where $n > 1$, we delete this rule and add new ones $A \to v_1 X_1$, $X_1 \to v_2 X_2$, $\ldots$ $X_{n-1} \to v_n B$, where $X_1, X_2, \ldots X_{n-1}$ are new variables added to G'. Thus we can delete all such rules $A \to \alpha B$ and similarly all rules $A \to \alpha$, where $|\alpha| > 1$.

Second, any rule of the form $A \to eA$ can be dropped without replacement, even if the rule is $S \to eS$, since such rules have no effect on a derivation. If $A \neq S$ and the rule $A \to eB$ occurs in P' for some $B \neq A$, it too can be deleted, provided every occurrence of A in the remaining rules is replaced by B. It should be clear that if $S \to^* w_1 w_2 \ldots w_n$ in G' before these changes and A occurs in this derivation, then all instances of A's in the derivation are replaced by B's and the derivation still holds. Cycles of such rules $A \to eB$, $B \to eC$, $C \to eA$, and so on will cause no difficulties. If rules of the form $S \to eB$ occur, they can be deleted similarly, with B becoming the new start symbol.

All the remaining rules are of the form $A \to vB$, $A \to v$, or $A \to e$. Any rule $A \to e$, where A is a variable which never appears on the right-hand side of some other rule, can be dropped, since such a rule can never occur in a derivation; the rule $S \to e$ can be allowed to remain. If $A \to e$ is a rule in P' and A does appear on the right-hand side of some other rule $B \to vA$, we can delete $A \to e$ from P' and add $B \to v$.

These changes in production rules clearly give a regular grammar, and one which is equivalent to G'. Thus $\mathcal{L}'$ is regular.

The proof for context-free languages is similar. The rules we will obtain using the technique in Theorem 4.4 will have the form $A \to BC$ or $A \to \alpha$, which are context-free rules, with the exception of the rules $A \to e$. Hopcroft and Ullman prove [pp. 62-63] that these so-called e-rules

can be added to context-free grammars without destroying the context-free property.

Theorem 4.10 Let $\mathcal{L}$ be context-sensitive, let $\mathcal{F}$ be a GFAO, and let $\mathcal{L}' = \mathcal{F}(\mathcal{L})$. Then $\mathcal{L}'$ will be context-sensitive, provided the following condition is met: there is an integer $n > 0$ such that whenever $\alpha \in \mathcal{L}'$, there is a $\beta \in \mathcal{L}$ such that $\alpha \in \mathcal{F}(\beta)$ and $|\beta| < n|\alpha|$.

Proof: The condition amounts to a linear bound on possible pre-images for words in $\mathcal{L}'$. An LBA which recognises words in $\mathcal{L}'$ would operate in a manner similar to that described in the proof of Theorem 4.5, except that all words must be checked having length up to n times the length of the input.

Corollary 4.11 The GRT of a Type 0 language is Type 0.

Proof: The method employed in Corollary 4.6 generalizes to GRT's.

These theorems indicate that the tricks used on p. 46 are not necessary to deal with various coding procedures which are similar to that used in Section 3. For example, to transform

A'A'''#A''''A'#A'''A''''

into

10111#111101#11101111

we can use a GFAO whose transition function is defined by

$\delta(q_o,A) = (q_1,e)$

$\delta(q_1,') = (q_1,1)$

$\delta(q_1,A) = (q_1,0)$

$\delta(q_1,\#) = (q_o,\#)$

Notice that the linear bound requirement imposed by Theorem 4.10 is met, with n = 2. A GFAO which "reverses" the above example would have transition function δ' defined by

$\delta'(q_o,1) = (q_1,A')$

$\delta'(q_1,1) = (q_1,')$

$\delta'(q_1,0) = (q_1,A)$

$\delta'(q_1,\#) = (q_o,\#)$.

For the results in Section 3 to apply fully to a new coding method, we must have a GRT of the original language into the new one and conversely. We have already seen that the GRT of a non-regular language is not necessarily non-regular. However the existence of the two GFAO's mentioned above is sufficient to prove the following:

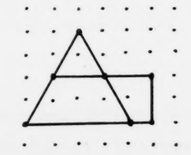
Theorem 4.12 All theorems proved in Section 3 remain true when our original coding procedure is replaced with the procedure described in ** on p. 46.

SECTION 5: FURTHER INVESTIGATIONS

The most important remaining problem is that mentioned at the end of Section 1, namely the identification or characterization of the class of geometrically non-planar figures. Of course a simple linguistic characterization for this class would be ideal, but this is perhaps more than we can reasonably expect. As one sees in Section 3, it becomes more apparent that, as Type 1 and even Type 0 languages begin to play the central role, there is a shift from linguistics to a more mechanical or computational aspect. In this sense, then, a linguistic treatment may not be entirely satisfactory. The following theorems are an illustration.

Theorem 5.1 The set of planar codings is recursively enumerable, that is, the language of planar codings is Type 0.

Proof: We will describe a procedure for checking codings which will tell us when a coding is planar. If we are given a coding c with n distinct vertices, we look at the set of points in the Cartesian plane with coordinates (x,y) where x = 0, 1, 2, ... n and y = 0, 1, 2, ... n. These points form an n × n grid. We now assign coordinates from this set to the vertices in c, trying to find a figure which is a realization of c. If a realization for c is found, then c is planar. If all possible ways of assigning coordinates fail to produce a realization for



A realization of c = {ABC, ADE, EF, CGD, BGF} on a 6 × 6 grid.

c, we try an n+1 × n+1 grid. By continuing in this manner, we will find a realization for c if one exists. If c is non-planar, this procedure never terminates.

To see that this procedure does terminate when c is planar, we note that if c is planar, then certainly some realization for c can be found in the Cartesian plane, in which the vertices have rational coordinates. This follows partly from the fact that when a line segment has rational endpoints, then any interior point which has one rational coordinate, has both coordinates rational. Further, we can "shrink" this realization so that it lies within the unit square bounded by (0,0), (0,1), (1,0), (1,1), and such that its vertices still have rational coordinates. Let n be the least common denominator for all these rational coordinates: that is, each coordinate can be expressed as x/n, where $0 \leq x \leq n$. Thus each vertex v_i has coordinates $(x_i/n, y_i/n)$, and hence we can consider c to have a realization on an n × n grid, where v_i's coordinates are (x_i, y_i).

For each coding c there is an integer m_c which has the property that if no realization for the coding can be found using an $m_c \times m_c$ grid, then no realization exists. The problem lies in determining what this integer is for a general coding. If, for example, it could be shown that there is a linear function f with the property that $f(|c|) \geq (m_c)^2$, then the language of planar codings would be Type 1.

Theorem 5.2 The class of GNP codings is recursive.

Proof: In 1930 Alfred Tarski gave a decision procedure which applies to sentences in the elementary theory of Euclidean geometry, using the

notions of equality, betweenness and equidistance. The geometric system used in this paper is a subsystem of Euclidean geometry, so Tarski's algorithm will apply.[6] To use the algorithm, we take a coding c and write a sentence which expresses all equality and betweenness relations holding for vertices in c, including statements about line segments in c which do not intersect one another. The algorithm will transform this into a sentence of elementary algebra involving real polynomials. The truth of this sentence is decided using powerful techniques from real analysis. As an example of the sentence we derive from a coding to apply the algorithm, consider the GNP coding {ABC, AED, BD, CE} given on p. 11. The description of this figure (coding) would be:

$$(\exists A)(\exists B)(\exists C)(\exists D)(\exists E)\{(A\neq B)\ \&\ (A\neq C)\ \&\ (A\neq D)\ \&\ (A\neq E)\ \&\ (B\neq C)\ \&\ (B\neq D)\ \&\ (B\neq E)\ \&\ (C\neq D)\ \&\ (C\neq E)\ \&\ (D\neq E)\ \&\ b(A,B,C)\ \&\ b(A,E,D)\ \&\ (\forall X)[b(B,X,E) \Rightarrow \text{not-}b(C,X,D)]\}.$$

A most promising area for further study is graph theory. Little has been done in graph theory with straight-line graphs, but interest is picking up, especially in connection with the four-color problem. In 1948 the graph theorist Fáry proved that any planar graph[7] can be realized in the plane using straight lines. This means, for example, that the coding mentioned above is realizable using straight lines, provided the vertices B and E are no longer considered to be interior vertices but endpoints. Thus the coding {AB, BC, AE, ED, BD, CE} is planar. It is worth outlining the proof of this theorem.

The proof uses induction on the number of vertices in the graph.

---

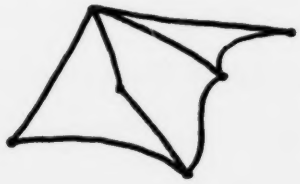
[6]We are indebted to Professor Haim Gaifman for pointing out that Tarski's algorithm applies in this situation.

[7]By a "graph" we will mean a coding in which every line segment or "edge" has length 2, and such edges need no longer be straight lines.

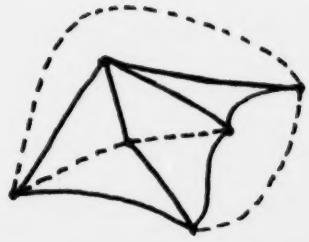
Suppose the result is true for graphs having n or fewer vertices and assume G to be a maximal planar graph with n+1 vertices. This means that no unconnected pair of vertices can be connected without making G non-planar. Since any planar graph with n+1 vertices is contained within such a maximal graph, it will suffice to find a straight-line representation for G. An interior vertex v is selected, and the vertices $v_1, v_2, \ldots v_m$ which are adjacent to v are located and listed in cyclic order. We can assume $m > 3$. It follows from maximality that there is a minimal path enclosing v having edges $v_1v_2, v_2v_3, \ldots v_mv_1$. We can remove vertex v together with the edges connecting it to the adjacent vertices, and add edges $v_1v_3, v_1v_4, \ldots v_1v_{m-1}$ if not already present. The resulting graph is maximal planar and has n vertices, so we can find a straight-line representation for it. The construction of this graph makes it possible to reinsert the vertex v inside the path $v_1v_2\ldots v_m$ in such a way that v can be connected to each v_i using straight-lines.[8]

Here is an example showing an application of the theorem:

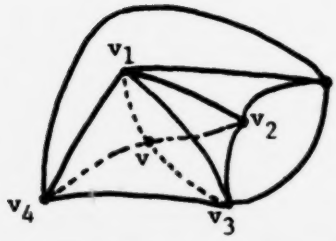
A: Graph with 6 vertices.



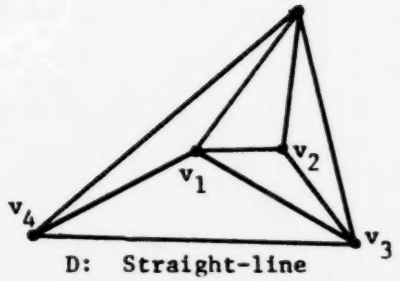
B: Maximal graph containing A.



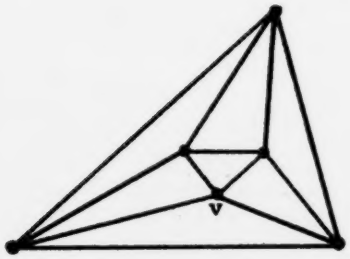
[8]For additional details, see Ore, The Four Color Problem, [pp. 5-8].



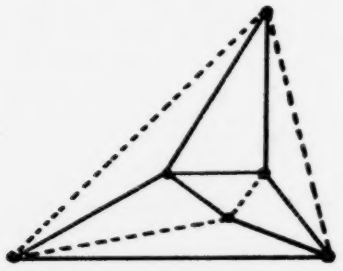
C: Select and remove vertex v and draw edge v_1v_3.



D: Straight-line representation for C.



E: Vertex v is reinserted.



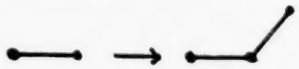
F: Remove certain edges to obtain straight-line representation for A.

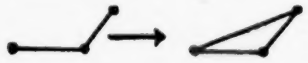
The reason for devoting so much space to this theorem and its proof is that we can apply it to a particular grammar which generates a class of planar straight-line graphs. We give a collection of production rules which allow us to proceed from a given coding to a larger one; these codings will be expressed here as sets of line segments as in Section 1. Our starting point is S = {AB}. The rules are:

1) If $XY \in T$ or $YX \in T$ and Z is a vertex not appearing in T, then from T we derive $T \cup \{YZ\}$ and also from T we derive $(T - \{XY\}) \cup \{XZ, ZY\}$.

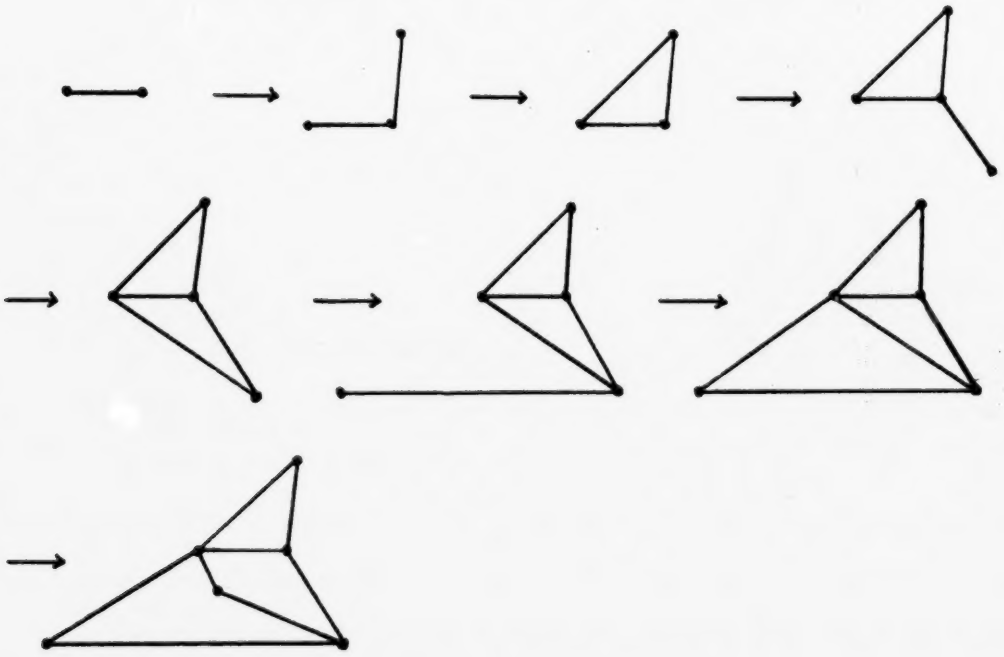
2) If $XY \in T$ and $YZ \in T$ then from T we can derive $T \cup \{XZ\}$, provided this latter coding contains neither Kuratowski subgraph.

One suggestive way of visualizing these rules is the following: our start symbol is " ⟷ " and the production rules correspond to these:

1) 

2) 

The figure A on p.54 might be "derived" as follows:

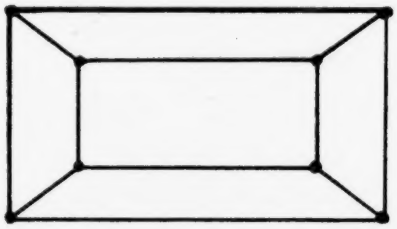


and a corresponding sequence of codings would be:

S = {AB} → {AB, BC} → {AB, BC, CA} → {AB, BC, CA, BD} → {AB, BC, CA, BD, DA} → {AB, BC, CA, BD, DA, DE} → {AB, BC, CA, BD, DA, DE, EA} → {AB, BC, CA, BD, DE, EA, DF, FA}.

The first thing we notice about these rules is that they seem to bear no relation to the rules of a formal grammar, which cannot be used for set generation. This is compensated for in their intuitive clarity, for it is true that the language which they generate is context-sensitive, and one can only imagine how intuitive the rules for a formal grammar

for the same language would be. As for the class of graphs, or rather the class of codings for graphs, which is generated, we can apply Fáry's theorem to show that any maximal planar graph can be so derived. Note that the production rules given are specially suited to the generation of triangles, and that any maximal planar graph is "triangulated", that is, each face is bounded by a path of length 3 (if not, we could draw a diagonal which bisects the face, contradicting maximality). However, it is not possible to generate all planar graphs with these rules; here is an example:



The last rule used in generating this graph cannot be rule 1, for this results in a vertex of order one or two (that is, with one or two adjacent vertices), and cannot be rule 2, for this rule completes a triangle, and the graph contains no triangles.

The preceding example indicates the direction in which notions of grammar and formal language must be extended for application to geometry and similar areas. Work is currently being done on "indexed" languages, in which the introduction of new vertex-labels in a coding can be handled more naturally, and on "graph" grammars, which are quite similar to the above example. Applications for such grammars are being increasingly found in the areas of picture processing and pattern recognition.

BIBLIOGRAPHY

Berge, Claude. The Theory of Graphs. London: Methuen & Company Ltd, 1966.

Chomsky, Noam. Aspects of the Theory of Syntax. Cambridge: MIT Press, 1969.

Chomsky, Noam. Cartesian Linguistics. New York: Harper & Row, 1966.

Chomsky, Noam, and Miller, George A. "Introduction to the Formal Analysis of Natural Languages," Handbook of Mathematical Psychology, Volume II, New York: John Wiley and Sons, 1967.

Ginsburg, Seymour. The Mathematical Theory of Context-free Languages. New York: McGraw-Hill, 1966.

Harary, Frank. Graph Theory. Reading, Massachusetts: Addison-Wesley, 1969.

Hopcroft, John E., and Ullman, Jeffrey D. Formal Languages and Their Relation to Automata. Reading, Massachusetts: Addison-Wesley, 1969.

Minsky, Marvin, and Papert, Seymour. Perceptrons. Cambridge: MIT Press, 1969.

Ore, Oystein. Graphs and Their Uses. Singer New Mathematical Library (New York: Random House), 1963.

Ore, Oystein. The Four Color Problem. New York: Academic Press, 1967.

Pavlidis, T. "Linear and Context-free Graph Grammars," Journal of the Association for Computing Machinery, Vol 19, No. 1, January 1972, pp. 11-22.

Rottmayer, William Arthur. "A Formal Theory of Perception," Institute for Mathematical Studies in the Social Sciences, Technical Report No. 161. Stanford University, 1970.

Shaw, Alan C. "A Formal Picture Description Scheme as a Basis for Picture Processing Systems," Information and Control, Vol 14, 1969, pp. 9-52.

Suppes, Patrick. "Stimulus-Response Theory of Finite A mata," Journal of Mathematical Psychology, Vol 6, No. 3, 1969, pp. 327-355.

Tarski, Alfred. "A Decision Method for Elementary Algebra and Geometry," The RAND Corporation, 1948, Report R-109.

Tutte, W. T. "How to Draw a Graph," Proc. London Math. Soc., Vol 3, No. 13, 1963, pp. 743-768.

(Continued from inside front cover)

96 R. C. Atkinson, J. W. Brelsford, and R. M. Shiffrin. Multi-process models for memory with applications to a continuous presentation task. April 13, 1966. (J. math. Psychol., 1967, 4, 277-300).

97 P. Suppes and E. Crothers. Some remarks on stimulus-response theories of language learning. June 12, 1966.

98 R. Bjork. All-or-none subprocesses in the learning of complex sequences. (J. math. Psychol., 1968, 1, 182-195).

99 E. Gammon. The statistical determination of linguistic units. July 1, 1966.

100 P. Suppes, L. Hyman, and M. Jerman. Linear structural models for response and latency performance in arithmetic. (In J. P. Hill (ed.), Minnesota Symposia on Child Psychology. Minneapolis, Minn.: 1967. Pp. 160-200).

101 J. L. Young. Effects of intervals between reinforcements and test trials in paired-associate learning. August 1, 1966.

102 H. A. Wilson. An investigation of linguistic unit size in memory processes. August 3, 1966.

103 J. T. Townsend. Choice behavior in a cued-recognition task. August 8, 1966.

104 W. H. Batchelder. A mathematical analysis of multi-level verbal learning. August 9, 1966.

105 H. A. Taylor. The observing response in a cued psychophysical task. August 10, 1966.

106 R. A. Bjork. Learning and short-term retention of paired associates in relation to specific sequences of interpresentation intervals. August 11, 1966.

107 R. C. Atkinson and R. M. Shiffrin. Some Two-process models for memory. September 30, 1966.

108 P. Suppes and C. Ihrke. Accelerated program in elementary-school mathematics--the third year. January 30, 1967.

109 P. Suppes and I. Rosenthal-Hill. Concept formation by kindergarten children in a card-sorting task. February 27, 1967.

110 R. C. Atkinson and R. M. Shiffrin. Human memory: a proposed system and its control processes. March 21, 1967.

111 Theodore S. Rodgers. Linguistic considerations in the design of the Stanford computer-based curriculum in initial reading. June 1, 1967.

112 Jack M. Knutson. Spelling drills using a computer-assisted instructional system. June 30, 1967.

113 R. C. Atkinson. Instruction in initial reading under computer control: the Stanford Project. July 14, 1967.

114 J. W. Brelsford, Jr. and R. C. Atkinson. Recall of paired-associates as a function of overt and covert rehearsal procedures. July 21, 1967.

115 J. H. Stelzer. Some results concerning subjective probability structures with semiorders. August 1, 1967

116 D. E. Rumelhart. The effects of interpresentation intervals on performance in a continuous paired-associate task. August 11, 1967.

117 E. J. Fishman, L. Keller, and R. E. Atkinson. Massed vs. distributed practice in computerized spelling drills. August 18, 1967.

118 G. J. Groen. An investigation of some counting algorithms for simple addition problems. August 21, 1967.

119 H. A. Wilson and R. C. Atkinson. Computer-based instruction in initial reading: a progress report on the Stanford Project. August 25, 1967.

120 F. S. Roberts and P. Suppes. Some problems in the geometry of visual perception. August 31, 1967. (Synthese, 1967, 17, 173-201)

121 D. Jamison. Bayesian decisions under total and partial ignorance. D. Jamison and J. Kozielecki. Subjective probabilities under total uncertainty. September 4, 1967.

122 R. C. Atkinson. Computerized instruction and the learning process. September 15, 1967.

123 W. K. Estes. Outline of a theory of punishment. October 1, 1967.

124 T. S. Rodgers. Measuring vocabulary difficulty: An analysis of item variables in learning Russian-English and Japanese-English vocabulary parts. December 18, 1967.

125 W. K. Estes. Reinforcement in human learning. December 20, 1967.

126 G. L. Wolford, D. L. Wessel, W. K. Estes. Further evidence concerning scanning and sampling assumptions of visual detection models. January 31, 1968.

127 R. C. Atkinson and R. M. Shiffrin. Some speculations on storage and retrieval processes in long-term memory. February 2, 1968.

128 John Holmgren. Visual detection with imperfect recognition. March 29, 1968.

129 Lucille B. Mlodnosky. The Frostig and the Bender Gestalt as predictors of reading achievement. April 12, 1968.

130 P. Suppes. Some theoretical models for mathematics learning. April 15, 1968. (Journal of Research and Development in Education, 1967, 1, 5-22)

131 G. M. Olson. Learning and retention in a continuous recognition task. May 15, 1968.

132 Ruth Norene Hartley. An investigation of list types and cues to facilitate initial reading vocabulary acquisition. May 29, 1968.

133 P. Suppes. Stimulus-response theory of finite automata. June 19, 1968.

134 N. Moler and P. Suppes. Quantifier-free axioms for constructive plane geometry. June 20, 1968. (In J. C. H. Gerretsen and F. Oort (Eds.), Compositio Mathematica. Vol. 20. Groningen, The Netherlands: Wolters-Noordhoff, 1968. Pp. 143-152.)

135 W. K. Estes and D. P. Horst. Latency as a function of number or response alternatives in paired-associate learning. July 1, 1968.

136 M. Schlag-Rey and P. Suppes. High-order dimensions in concept identification. July 2, 1968. (Psychom. Sci., 1968, 11, 141-142)

137 R. M. Shiffrin. Search and retrieval processes in long-term memory. August 15, 1968.

138 R. D. Freund, G. R. Loftus, and R.C. Atkinson. Applications of multiprocess models for memory to continuous recognition tasks. December 18, 1968.

139 R. C. Atkinson. Information delay in human learning. December 18, 1968.

140 R. C. Atkinson, J. E. Holmgren, and J. F. Juola. Processing time as influenced by the number of elements in the visual display. March 14, 1969.

141 P. Suppes, E. F. Loftus, and M. Jerman. Problem-solving on a computer-based teletype. March 25, 1969.

142 P. Suppes and Mona Morningstar. Evaluation of three computer-assisted instruction programs. May 2, 1969.

143 P. Suppes. On the problems of using mathematics in the development of the social sciences. May 12, 1969.

144 Z. Domotor. Probabilistic relational structures and their applications. May 14, 1969.

145 R. C. Atkinson and T. D. Wickens. Human memory and the concept of reinforcement. May 20, 1969.

146 R. J. Titiev. Some model-theoretic results in measurement theory. May 22, 1969.

147 P. Suppes. Measurement: Problems of theory and application. June 12, 1969.

148 P. Suppes and C. Ihrke. Accelerated program in elementary-school mathematics--the fourth year. August 7, 1969.

149 D. Rundus and R.C. Atkinson. Rehearsal in free recall: A procedure for direct observation. August 12, 1969.

150 P. Suppes and S. Feldman. Young children's comprehension of logical connectives. October 15, 1969.

(Continued on back cover)

(Continued from inside back cover)

151 Joaquim H. Laubsch. An adaptive teaching system for optimal item allocation. November 14, 1969.
152 Roberta L. Klatzky and Richard C. Atkinson. Memory scans based on alternative test stimulus representations. November 25, 1969.
153 John E. Holmgren. Response latency as an indicant of information processing in visual search tasks. March 16, 1970.
154 Patrick Suppes. Probabilistic grammars for natural languages. May 15, 1970.
155 E. Gammon. A syntactical analysis of some first-grade readers. June 22, 1970.
156 Kenneth N. Wexler. An automaton analysis of the learning of a miniature system of Japanese. July 24, 1970.
157 R. C. Atkinson and J.A. Paulson. An approach to the psychology of instruction. August 14, 1970.
158 R.C. Atkinson, J.D. Fletcher, H.C. Chetin, and C.M. Stauffer. Instruction in initial reading under computer control: the Stanford project. August 13, 1970.
159 Dewey J. Rundus. An analysis of rehearsal processes in free recall. August 21, 1970.
160 R.L. Klatzky, J.F. Juola, and R.C. Atkinson. Test stimulus representation and experimental context effects in memory scanning.
161 William A. Rottmayer. A formal theory of perception. November 13, 1970.
162 Elizabeth Jane Fishman Loftus. An analysis of the structural variables that determine problem-solving difficulty on a computer-based teletype. December 18, 1970.
163 Joseph A. Van Campen. Towards the automatic generation of programmed foreign-language instructional materials. January 11, 1971.
164 Jamesine Friend and R.C. Atkinson. Computer-assisted instruction in programming: AID. January 25, 1971.
165 Lawrence James Hubert. A formal model for the perceptual processing of geometric configurations. February 19, 1971.
166 J. F. Juola, I.S. Fischler, C.T. Wood, and R.C. Atkinson. Recognition time for information stored in long-term memory.
167 R.L. Klatzky and R.C. Atkinson. Specialization of the cerebral hemispheres in scanning for information in short-term memory.
168 J.D. Fletcher and R.C. Atkinson. An evaluation of the Stanford CAI program in initial reading (grades K through 3). March 12, 1971.
169 James F. Juola and R.C. Atkinson. Memory scanning for words versus categories.
170 Ira S. Fischler and James F. Juola. Effects of repeated tests on recognition time for information in long-term memory.
171 Patrick Suppes. Semantics of context-free fragments of natural languages. March 30, 1971.
172 Jamesine Friend. Instruct coders' manual. May 1, 1971.
173 R.C. Atkinson and R.M. Shiffrin. The control processes of short-term memory. April 19, 1971.
174 Patrick Suppes. Computer-assisted instruction at Stanford. May 19, 1971.
175 D. Jamison, J.D. Fletcher, P. Suppes, and R.C. Atkinson. Cost and performance of computer-assisted instruction for compensatory education.
176 Joseph Offir. Some mathematical models of individual differences in learning and performance. June 28, 1971.
177 Richard C. Atkinson and James F. Juola. Factors influencing speed and accuracy of word recognition. August 12, 1971.
178 P. Suppes, A. Goldberg, G. Kanz, B. Searle, and C. Stauffer. Teacher's handbook for CAI courses. September 1, 1971.
179 Adele Goldberg. A generalized instructional system for elementary mathematical logic. October 11, 1971.
180 Max Jerman. Instruction in problem solving and an analysis of structural variables that contribute to problem-solving difficulty. November 12, 1971.
181 Patrick Suppes. On the grammar and model-theoretic semantics of children's noun phrases. November 29, 1971.
182 Georg Kreisel. Five notes on the application of proof theory to computer science. December 10, 1971.
183 James Michael Moloney. An investigation of college student performance on a logic curriculum in a computer-assisted instruction setting. January 28, 1972.
184 J.E. Friend, J.D. Fletcher, and R.C. Atkinson. Student performance in computer-assisted instruction in programming. May 10, 1972.
185 Robert Lawrence Smith, Jr. The syntax and semantics of ERICA. June 14, 1972.
186 Adele Goldberg and Patrick Suppes. A computer-assisted instruction program for exercises on finding axioms. June 23, 1972.
187 Richard C. Atkinson. Ingredients for a theory of instruction. June 26, 1972.
188 John D. Bonvillian and Veda R. Charrow. Psycholinguistic implications of deafness: A review. July 14, 1972.
189 Phipps Arabie and Scott A. Boorman. Multidimensional scaling of measures of distance between partitions. July 26, 1972.
190 John Ball and Dean Jamison. Computer-assisted instruction for dispersed populations: Systems models. September 15, 1972.
191 W. R. Sanders and J. R. Ball. Logic documentation standard for the Institute for Mathematical Studies in the Social Sciences. October 4, 1972.
192 M.T. Kane. Variability in the proof behavior of college students in a CAI course in logic as a function of problem characteristics. October 6, 1972.
193 P. Suppes. Facts and fantasies of education. October 18, 1972.
194 R. C. Atkinson and J. F. Juola. Search and decision processes in recognition memory. October 27, 1972.
195 P. Suppes, R. Smith, and M. Léveillé. The French syntax and semantics of PHILIPPE, part 1 : Noun phrases. November 3, 1972.
196 D. Jamison, P. Suppes, and S. Wells. The effectiveness of alternative instructional methods: A survey. November 1972.
197 P. Suppes. A survey of cognition in handicapped children. December 29, 1972.
198 B. Searle, P. Lorton, Jr., A. Goldberg, P. Suppes, N. Ledet, and C. Jones. Computer-assisted instruction program: Tennessee State University. February 14, 1973.
199 D. R. Levine. Computer-based analytic grading for German grammar instruction. March 16, 1973.
200 P. Suppes, J.D. Fletcher, M. Zanotti, P. V. Lorton, Jr., and B. W. Searle. Evaluation of computer-assisted instruction in elementary mathematics for hearing-impaired students. March 17, 1973.
201 G. A. Huff. Geometry and formal linguistics. April 27, 1973.